Published by IOP Publishing for Sissa Medialab

Received: *February 28, 2016*
Accepted: *May 15, 2016*
Published: *June 14, 2016*

2016 JINST 11 P06008

The ATLAS Data Acquisition and High Level Trigger system

The ATLAS TDAQ Collaboration

E-mail: i73@nikhef.nl

Abstract: This paper describes the data acquisition and high level trigger system of the ATLAS experiment at the Large Hadron Collider at CERN, as deployed during Run 1. Data flow as well as control, configuration and monitoring aspects are addressed. An overview of the functionality of the system and of its performance is presented and design choices are discussed.

Keywords: Control and monitor systems online; Data acquisition concepts; Online farms and online filtering; Trigger concepts and systems (hardware and software)

doi:10.1088/1748-0221/11/06/P06008

Contents

2016 JINST 11 P06008

2016 JINST 11 P06008

2016 JINST 11 P06008

2016 JINST 11 P06008

1 Introduction

1.1 The LHC

The Large Hadron Collider (LHC) [1] at CERN, Geneva, Switzerland is a 27 km circumference synchrotron that can accelerate two counter-rotating beams of protons or heavy ions simultaneously. After acceleration the beams are kept circulating in the machine while colliding at four interaction points, for protons for a period of typically 10-20 hours. The design proton energy is 7 TeV. the LHC has been operated in 2010 and 2011 with collisions of 3.5 TeV protons and, for a limited time, also with lead-lead collisions (using lead nuclei with an energy of 2.76 TeV/Nucleon) [2]. In 2012 the proton energy has been increased to 4 TeV and recently, after the long shutdown of 2013 and 2014, to 6.5 TeV. Initially the maximum nominal instantaneous luminosity for proton-proton collisions was $10^{27}\ cm^{-2}\ s^{-1}$. The luminosity increased rapidly during 2010 and 2011, with more modest increases in 2012, so that by December 2012 the maximum attained was $7.7\ 10^{33}\ cm^{-2}\ s^{-1}$, approaching the design luminosity of $10^{34}\ cm^{-2}\ s^{-1}$.

1.2 The ATLAS detector

The ATLAS detector [3] surrounds interaction point 1 of the LHC, about 100 m below the surface and opposite to the main entrance of the CERN Meyrin site. ATLAS is designed for studying particles produced by proton-proton interactions, but is also used for studying heavy ion collisions. Figure 1 shows a view of the detector, with part of it removed to show parts otherwise hidden. A unique feature of the detector is the toroidal magnetic field around the outside of the detector, allowing high-precision measurement of muon momenta. It is generated by eight main superconducting coils, 25.3 m long, extending from a radius of 4.7 m to 10.1 m, in the central part of the detector.

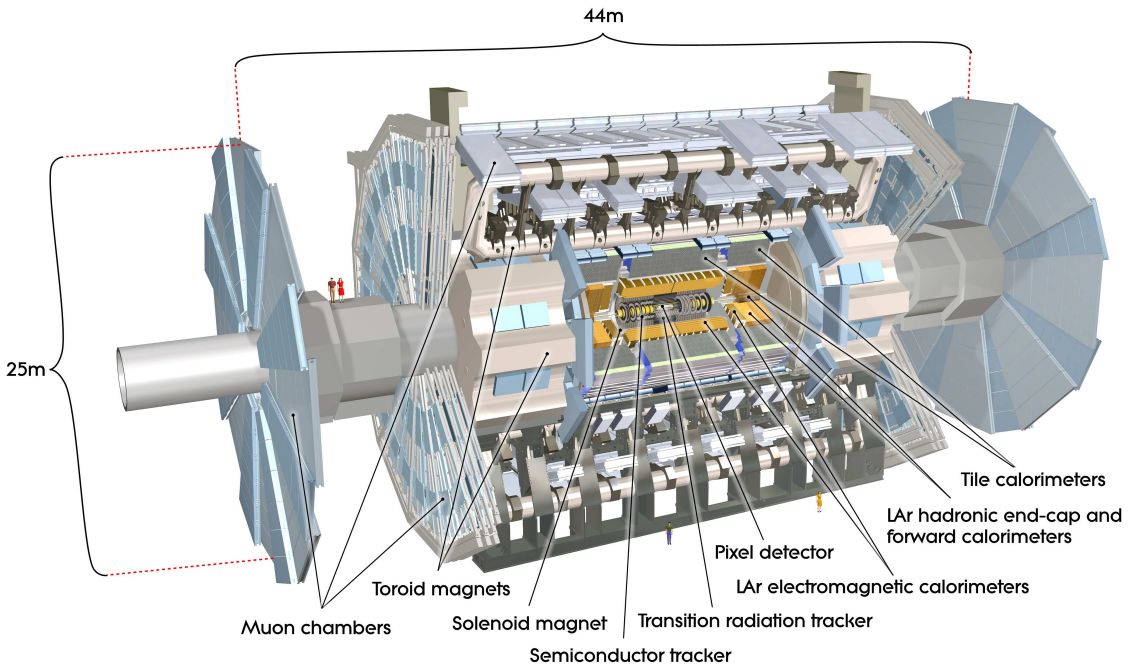


Figure 1. Cut-away view of the ATLAS detector.

Each of these coils is enclosed in its own vacuum tube. Plus at each end of the detector there is a large vacuum vessel containing eight smaller coils, each with a length of 5 m and extending from a radius of 82.5 cm to 5.35 m. The full name of the ATLAS experiment, "A Toroidal LHC ApparatuS", refers to the toroidal field.

The interaction point is at the centre of the detector. The detector itself has a layered structure. In the following a "sub-detector" refers to a part of the detector built using a single detector technology. In most cases a sub-detector consists of a barrel part and two or more end-cap parts. Going from the interaction point to the outside of the detector the sub-detectors first encountered are those forming the inner detector: the silicon pixel detector, the SemiConductor Tracker (SCT), built from silicon strip detectors, and the Transition Radiation Tracker (TRT), built from Polyimide drift tubes with 4 mm diameter and interleaved with fibers (barrel) or foils (end-caps) for generating transition radiation. The inner detector is enclosed by a superconducting solenoid generating a magnetic field of 2 T. The solenoid and the barrel liquid argon electromagnetic calorimeter surrounding it share the same vessel. Forward liquid argon calorimetry consists of electromagnetic as well as hadronic parts. The barrel hadronic calorimeter, surrounding the electromagnetic calorimeter, is an iron-scintillator assembly. It is known as the "tile calorimeter": scintillating tiles are read out using wave-length shifting optical fibers. For the muon spectrometer surrounding the calorimeters four different technologies have been used: for precision position measurements layers of drift tubes ("Monitored Drift Tubes" (MDTs) and in the end-caps Cathode Strip Chambers (CSCs), for triggering Resistive Plate Chambers (RPCs) and in the end-caps Thin Gap Chambers (TGCs)). The setup is complemented by several small detectors in the very forward directions (not shown in figure 1). Figure 2 shows an overview of the underground areas and surface buildings.

2016 JINST 11 P06008

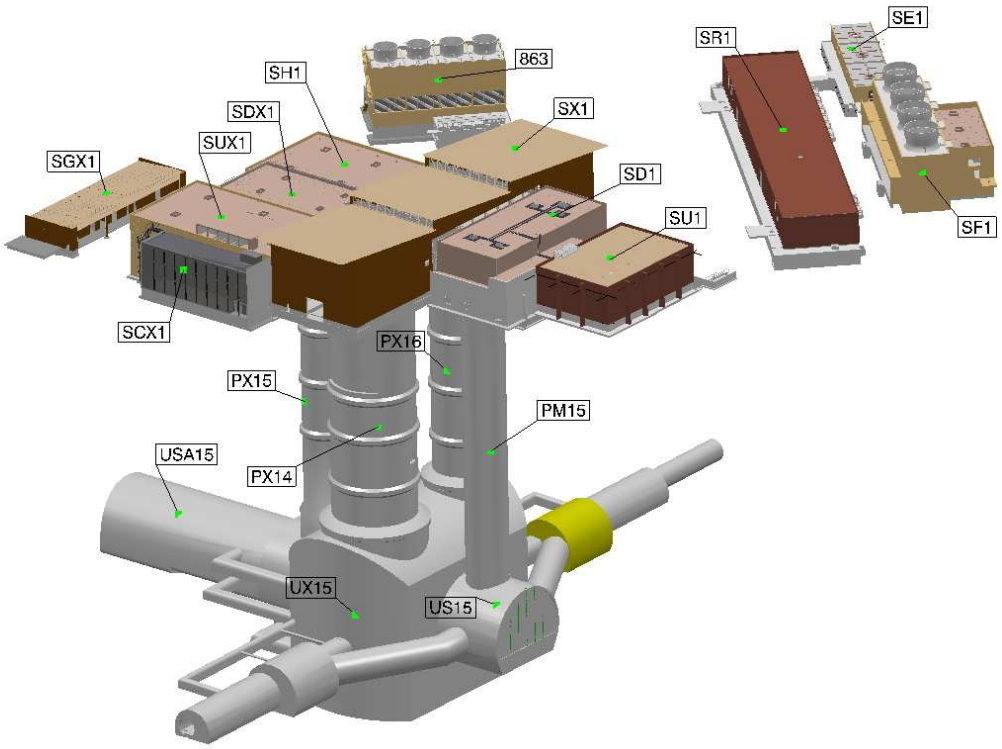


Figure 2. View of the ATLAS underground areas and surface buildings. The experiment is located in UX15, US15 and USA15 serve as counting rooms. A barrack located in SDX1 houses the high level trigger processors. The ATLAS control room is located at the ground floor of the SCX1 building.

1.3 Event selection: first and high level triggering

The beams of the LHC consist of trains of particle bunches [4]. The minimum time interval between passage of successive bunches within a train is 25 ns. Thus collisions can take place every 25 ns within a time interval determined by the lengths of the bunches, i.e. typically shorter than 1 ns. At an instantaneous luminosity of $10^{34}\ cm^{-2}\ s^{-1}$ and bunch spacing of 25 ns the average number of interactions is about 23 per bunch-crossing, corresponding to about 10^9 interactions per second.[1] Selective triggering is therefore required. Association of a unique bunch-crossing with each event is necessary to avoid background from collisions corresponding to other bunch-crossings. Furthermore, to avoid excessive dead time the trigger should be able to analyze event data at a rate of 40 MHz. ATLAS employs three levels of trigger to meet these requirements. The first level (L1) [4] is built from dedicated hardware and can analyze event data at the required rate of 40 MHz. This is achieved by making use of analog sums of calorimeter signals formed on the detector and of signals of dedicated muon trigger chambers (RPCs and TGCs). Consequently event selection is only possible on the basis of energy depositions in the calorimeters and of muon track segments. Figure 3 shows a schematic layout of the L1 trigger. It is located in the USA15 underground area, as close to the detector cavern as

[1] Except for a few test runs the bunch spacing was 50 ns for Run 1, at the highest luminosity this resulted in an average of 35 interactions per bunch-crossing.

2016 JINST 11 P06008

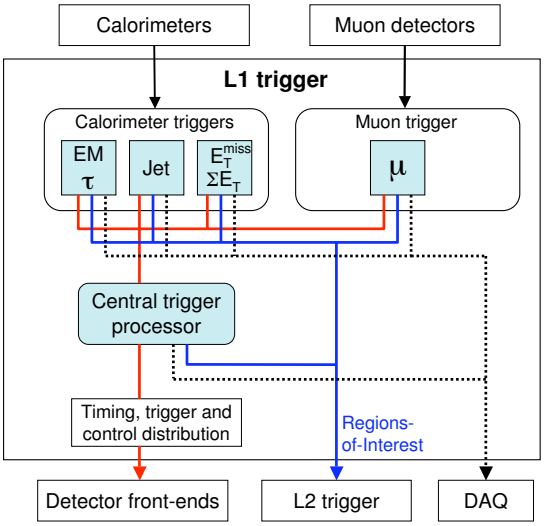


Figure 3. Block scheme of the first level trigger.

possible, to minimize the lengths of the cables used for forwarding the analog sums to the trigger and to minimize the time needed for sending the trigger accepts to the on-detector readout electronics.

By choosing appropriate thresholds the L1 trigger has been operated during Run 1 with a maximum accept rate of 60–65 kHz, somewhat lower than the maximum design rate of 75 kHz, to prevent excessive dead time. The readout of the detector has been upgraded during the long shutdown of 2013 and 2014 to allow for 100 kHz accept rate. The L1 trigger can handle an input rate equal to the maximum bunch-crossing rate of 40 MHz. Its maximum latency is about 2.5 μs, i.e. smaller than the maximum of about 3 μs imposed by the depth of the on-detector buffer memories. This latency includes the transit times of signals between detectors and trigger system and the time required for sending the trigger accepts to the on-detector readout electronics. Data corresponding to events accepted by L1 are further analyzed by software running in computer farms to provide two further levels of triggering. The second level (L2) makes use of a fraction of the full precision detector data and reduces the rate further. The original design aimed for 3.5 kHz, although during Run 1 a maximum rate of about 5–6 kHz was allowed. The design value of the output rate of the last trigger level, which has been given the name "Event Filter" (EF), is about 200–300 Hz, during Run 1 the maximum output rate was about twice as high. The two levels of the software trigger are collectively known as the High Level Trigger (HLT).

L1 accept decisions are distributed via the TTC (Timing, Trigger and Control) system [5–7] to the readout electronics, on-detector as well as off-detector, see figure 4. The Central Trigger Processor (CTP) of the first level trigger receives from the RF2TTC interface [4, 8] three clock signals with a frequency equal to 3564 times the revolution frequency of a bunch of 11.2 kHz, i.e. 40.078 MHz (one clock signal for each beam and one clock signal equal to the maximum collision rate), and two clock signals with a frequency equal to the revolution frequency. The CTP uses the LHC clock signal as clock for sending information via the Local Trigger Processor (LTP) modules [9], TTC-VME (TTCvi) modules [10] and TTCex laser transmitters [11].

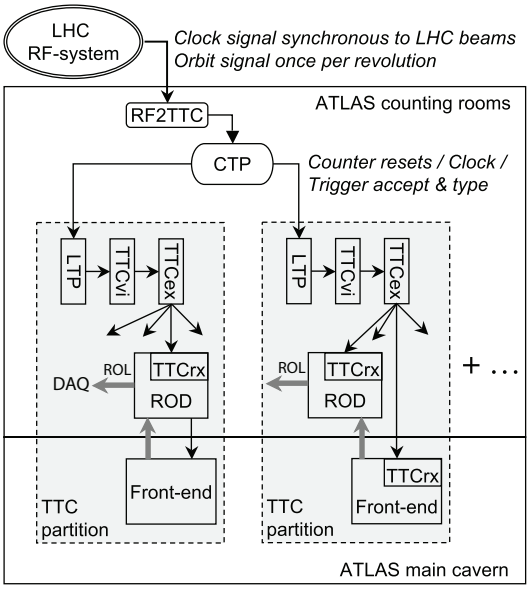


Figure 4. Overview of generation and distribution of timing and trigger signals by the Timing Trigger and Control (TTC) system and of the readout of the detector.

1.4 Readout

As illustrated in figure 4 the TTC information is received either by the front-end electronics directly via TTC receiver ASICs (TTCRx [12]), for examples see refs. [13] (LAr calorimeters) or [14] (MDTs), or indirectly via the ReadOut Drivers (RODs), for examples see refs. [15] (Pixels) or [16] (TRT). The RODs of the sub-detectors of which the front-end electronics connect directly to the TTC system also receive the TTC information. The RODs connect via the ReadOut Links (ROLs) to the DAQ (Data AcQuisition) system. Data are pushed from the front-end electronics upon the arrival of L1 accepts into the RODs and then forwarded via the ROLs. The L1 accept signals are accurately timed with respect to the associated bunch-crossings to facilitate reading out of data corresponding to the correct bunch-crossing. As indicated in the figure the TTC system is subdivided into "TTC partitions". For test and calibration purposes these partitions can be operated in parallel, with the LTP modules generating triggers instead of the CTP. The buffers of the DAQ system are grouped in the same way as the RODs from which they receive data. The rest of the DAQ system can be logically subdivided ("partitioned"), so that independent and simultaneous data acquisition for different TTC partitions is possible.

The RODs are sub-detector specific and custom built, in most cases in the form of 9U VME cards. The buffers of the DAQ system, the ReadOut Buffers (ROBs), are also custom built but do not have sub-detector specific functionality. The RODs are considered to be part of the sub-detector electronics, while the ROBs are part of the DAQ system. The links (ROLs) connecting RODs to ROBs make use of the S-link protocol [17, 18] and consist of optical fibers. Each ROB connects to a single ROL. For most sub-detectors the maximum throughput per link is 160 MB/s, as originally specified, but the ROBs can handle up to 200 MB/s. Data sent across the links are checked for transmission errors. By means of an XON-XOFF flow protocol data transmission is halted when

2016 JINST 11 P06008

a ROB cannot receive additional data. Table 1 provides an overview of the TTC partitions, the number of ROLs per partition, as well as the amount of data produced per partition.

For each L1 accept, information is output to the L2 system on "Regions of Interest" (RoIs) found in L1, i.e. geographical areas in the detector defined by the pseudorapidity η and the azimuthal angle ϕ of the objects which triggered L1. The L2 trigger subsequently requests the corresponding full precision data from the ROBs in which the data are stored. After analysis of the data received the trigger can also request additional data. Upon an accept of the L2 trigger, which also can be forced, e.g. for a calibration trigger, the Event Builder requests all event data from the ROBs and forwards these to the Event Filter. After acceptance the event is stored for further offline analysis. A so-called luminosity block is a set of events collected during a short time interval (1–2 minutes) for which the conditions for data taking were stable (approximately constant luminosity, no change in detector operating conditions). Together with the RoIs the luminosity block number, assigned by the L1 trigger, is also communicated to the L2 system, as trigger conditions may depend on it. The luminosity block number is stored in the event data forwarded to the Event Filter by the Event Builder.

An overview of the ATLAS electronics can be found in ref. [19].

1.5 Identification of events and data format

The front-end electronics send event data, via sub-detector specific links, to the RODs. The format and organization of these data are sub-detector specific as well. Event data are associated with an L1 identifier (L1Id) and a bunch-crossing identifier (BCId). At the start of a run all bits of the L1Id are set to 1. The L1Id is incremented upon reception of the L1 accept signal sent via the TTC system, so the L1Id of the first event in a run is 0. L1 accept (L1A) signals and messages are encoded by the TTC system using one of the LHC clock signals (by means of Biphase Mark encoding [5, 6, 12]). This clock is recovered by the TTC receiver ASICs and used for incrementing the BCId. The latter is reset to 0 after a Bunch Counter Reset (BCR) command is received, which is sent once per orbit period via the TTC system. The L1Id is reset to its start value (all bits 1) upon receipt of an Event Counter Reset (ECR) command. ECR commands are sent every few seconds to minimize the probability that incorrect L1Ids occur owing to missed L1A signals or, if this happens, to minimize the number of incorrect L1Ids. For each event the BCIds of all fragments should be identical, which allows a check of the correctness of the L1Ids.

The RODs assemble event fragments, each with a header and trailer as defined in ref. [20], from the data received from the front-end electronics. The header consists of the following nine 32-bit words: start of header marker, the size of the header (always 9), a number indicating the version of the data format of the fragment, an identifier of the ROD, the number of the run, the extended L1Id, the BCId, the L1 trigger type and finally a word reserved for sub-detector specific information. The extended L1Id stores in its least significant 24 bits the L1Id and in its 8 most significant bits the ECR identifier (ECRId), which is obtained by counting the ECR commands (starting from 0). The latter counting is done by the RODs. The counting of L1As and LHC clock cycles (for forming the BCIds) is done in the TTC receiver ASICs, but may also be done independently in the RODs and in the front-end electronics, which permits an additional check for incorrect L1Ids and BCIds in the RODs. Each event fragment may also contain status information, the last word of the three word trailer of each fragment indicates whether the status information precedes or follows the event data

2016 JINST 11 P06008

Table 1. Numbers of ROLs and readout PCs (ROS PCs, most PCs have 4 PCI custom plugin cards, each accommodating 3 ROBs) per detector TTC partition, as well as the observed (or expected) data size per L1 accept for luminosities of 3.5 10^{33} and 10^{34} cm^{-2} s^{-1} (in brackets) respectively (These luminosities correspond to 16.7 and 23 interactions per bunch-crossing, 50 and 25 ns bunch spacing, 7 and 14 TeV c.m. energy respectively).

TTC Partition			Number of ROLs	Number of ROS PCs	Data per L1 accept (kB)
Inner detector	Pixel	Layer 0	44	4	42 (60)
		Disks	24	2	
		Layers 1–2	64	6	
	SCT	End-cap A	23	2	64 (110)
		End-cap C	23	2	
		Barrel A	22	2	
		Barrel C	22	2	
	TRT	End-cap A	64	6	195 (307)
		End-cap C	64	6	
		Barrel A	32	3	
		Barrel C	32	3	
Calorimetry	LAr	EM barrel A	224	20	735 (576)
		EM barrel C	224	20	
		EM endcap A	138	12	
		EM endcap C	138	12	
		HEC	24	2	
		FCal	14	2	
	Tile	Barrel A	16	2	94 (48)
		Barrel C	17	2	
		Extended barrel A	16	2	
		Extended barrel C	16	2	
Muon spectrometer	MDT	Barrel A	50	4	83 (154)
		Barrel C	50	4	
		End-cap A	52	4	
		End-cap C	52	4	
	CSC	End-cap A	8	1	5 (10)
		End-cap C	8	1	
L1	Calorimeter	CP	12	2	30 (28) (can be varied)
		JEP	10	2	
		PP	32	3	
	Muon RPC	Barrel A	16	2	26 (12)
		Barrel C	16	2	
	Muon TGC	End-cap A	12	1	3 (6)
		End-cap C	12	1	
	MUCTPI		1	1	0.2 (0.1)
	CTP		1	1	0.7 (0.2)
Forward Detectors	BCM		3	1	1.6 (1)
	LUCID		1	1	0.1 (1)
	ALFA		2	1	only used in dedicated runs (1)
	ZDC		4	1	3.7 (1)
HLT		L2			22
		EF			50
		Total	1583	151	1377 (1311)

2016 JINST 11 P06008

(which varies according to the sub-detector). The first word of the trailer contains the number of status words, the second word the number of data words.

The ROD fragments are passed via the ReadOut Links (ROLs) to the DAQ system (to the ROBs). The RODs add control words indicating the beginning and the end of the fragment and a checksum to each fragment. These are discarded by the ROBs, after checking for errors. An additional header and trailer, with status words containing bits for signaling any errors found, are added to each ROD fragment by the ROBs. The contents of the ROD fragments are not altered by the DAQ system.

1.6 Overview of the contents of the next sections

The DAQ system and the HLT (High Level Trigger), consisting of the L2 trigger and the Event Filter, form the ATLAS DAQ/HLT system, the TDAQ (Trigger and Data AcQuisition) system includes also the L1 trigger. In the next sections the internal organization and the deployment of the DAQ/HLT system are described. An overview of the L1 trigger can be found in ref. [3], for more details see refs. [4, 21–28]. Section 2 focuses on hardware and software aspects of the systems. Section 3 contains an overview of results of performance tests and of observations from data taking, while in section 4 design and technology choices are discussed. In section 5 conclusions and an outlook are presented. Appendices contain details on hardware items, a short list of definitions and a list of acronyms. The nature of the trigger algorithms executed by the HLT, as well as their effectiveness with respect to background rejection and with respect to efficiency for acceptance of events with signatures of interest are not discussed in this paper, an overview is presented in ref. [29].

2 Description of the design and implementation of the DAQ/HLT system

2.1 Architecture and system components

2.1.1 Overview

The DAQ/HLT system interfaces to the detector readout and L1 trigger on the input side, and to the mass storage in the CERN computing centre on the output side. Event rates and data volumes observed during data taking in September 2011 and the expected values at the design luminosity of the LHC as specified in the ATLAS High-Level Trigger, Data-Acquisition and Controls Technical Design Report [30] are summarized in table 2. The output requirements are not only driven by the technical constraints on the DAQ side but, more importantly, by the capability of the CERN Tier-0 centre to store permanently the amount of data output, and of the world wide ATLAS Grid system to process and reprocess the data as required. A block scheme of the system is presented in figure 5.

The ATLAS trigger system reduces the event rate in a three level architecture (1.3). After an event has been accepted by the L1 trigger it is moved from the detector specific front-end buffers via the RODs into a common readout system (ROS) containing the ROBs (1.4). From here on the L2 trigger and the Event Builder have access to the data via an Ethernet based network.

The high level trigger (L2 and the Event Filter) is implemented in software running on server computers. To avoid building full events at the L1 accept rate of at maximum 75 kHz the L2 part of the HLT uses only a subset of the data. It is guided by information that is provided by the L1 muon and calorimeter systems in the form of co-ordinates of centres of areas in η/ϕ space where the L1 trigger has e.g. identified tracks in the muon system or clusters in the calorimeter. These areas are

2016 JINST 11 P06008

Table 2. Typical event rates and data volumes observed during data taking in September 2011 (for a fill of about 10 hours with peak luminosity of 3.3 10^{33} cm^{-2} s^{-1}) and expected values for design luminosity (10^{34} cm^{-2} s^{-1}) as presented in the ATLAS TDAQ Technical Design Report (TDR) [30] for a projected L1 accept rate of 100 kHz. The maximum L1 accept rate specified in the TDR is 75 kHz. Typically about 1/3 of the events written to storage are calibration events with a size smaller than 10% of the size of physics events.

	Input rate (2011)	Bandwidth (2011)	Input rate (TDR)	Bandwidth (TDR)
L2 (peak)	55 kHz	3 GB/s	75 (100) kHz	1.5 GB/s
Event Builder (peak)	5.5 kHz	8 GB/s	3.5 kHz	5.3 GB/s
Storage (average)	600 Hz	550 MB/s	200 Hz	300 MB/s

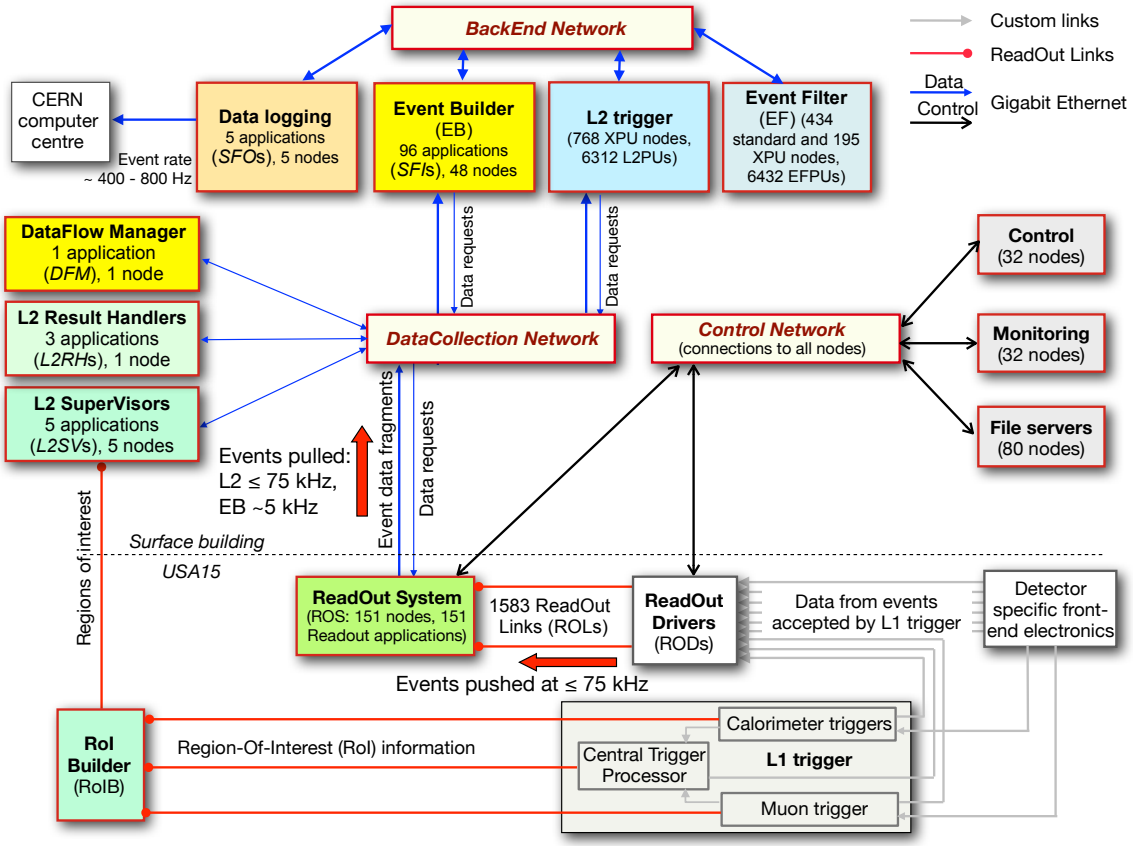


Figure 5. Block scheme of the Trigger and DAQ system. The numbers of nodes indicated are for the system as installed in September 2011, where either 1 or 4 nodes may be housed in a single chassis (appendix A). XPU nodes are nodes that can be used either for the L2 trigger or for Event Filter processing, L2PUs and EFPUs are applications executing the L2 and EF trigger algorithms respectively. For clarity only a few of the Control Network connections are shown.

2016 JINST 11 P06008

referred to as "Regions of Interest", abbreviated as "RoIs". The RoI Builder (2.4.1) combines the RoI information from various sources within the L1 trigger in real-time and makes it available to L2.

By requesting only RoI data (i.e. data from Regions of Interest) the bandwidth required for the L2 trigger is a fraction (a typical number being 5%) of the total bandwidth that would be needed for reading out the full event data.

After the L2 trigger has generated a decision the event is either discarded or built at the L2 accept rate. The full event data is passed to the Event Filter stage of the HLT, where predominantly offline algorithms are used for further event selection [29].

After the Event Filter has accepted an event its data are passed to the one of the data logging farm nodes running the Sub-Farm Output (SFO) application that stores the data on disk. The transfer to the CERN computer centre occurs asynchronously and independently from the status of the data acquisition. In case of external network failures the SFOs can buffer enough data on disk to keep the experiment running for at least 24 hours.

Most of the functionality of the DAQ/HLT system is provided by a set of different applications, running under the Linux operating system on PCs (high-end rack-mountable server machines). The newest machines consist of a chassis containing 4 independent computers. The computers are referred to as nodes, a node therefore does not refer to a chassis but to what can be defined as "an endpoint of a network running an operating system". Typically a number of instances of the same application are running in parallel on each node. In this context acronyms used in this paper refer to the applications and not to the nodes, e.g. L2RH refers to the L2 Result Handler application.

2.1.2 Control, configuration and monitoring

All applications are configured and controlled by a common software framework via a separate control network, which is also used for monitoring purposes.

The sequence of steps to start a run is governed by a common state machine that is implemented in all controlled applications (2.11).

For normal data taking the structure of the system and the settings of all necessary parameters are specified in a set of XML files, which combined form the configuration database (2.11.3) [30]. The configuration specified is referred to as the "ATLAS partition" and consists of a set of "partitions", which correspond to one or more TTC partitions. A partition contains one or more "segments": independently configurable and controllable parts of the TDAQ system. For testing or calibration individual partitions can be used independently of the rest of the TDAQ system. Dedicated test setups can be described in the same way as the TDAQ system, the configuration of such a test setup is also referred to as a "partition".

In addition to system configuration data trigger configuration data and so-called conditions data are used by several HLT components such as the selection algorithms (2.2.4). Conditions data describe the status of the detector at any given time and are stored in the conditions database. Each entry in this Oracle database has an interval of validity (IOV).

The common monitoring framework (2.12) permits the retrieval of event data in parallel to the normal data flow at various places in the DAQ system. The Information Service (IS) (2.2.2) and Online Histogram Service (OHS) (2.12.1) provide a common base used by all applications for publishing and retrieving real-time status information such as counters and histograms.

2016 JINST 11 P06008

2.1.3 Data flow

The event data are transferred over a dedicated network, the DataCollection Network (2.10.1), whose structure reflects the flow of data in the system.

The readout system (ROS) (2.3), the L2 system (2.4) and the Event Builder (2.5) are connected to two central core switches, the ROS and the L2 processing nodes via a layer of intermediate concentrator switches. The second central switch provides redundancy and additional bandwidth. After an event has been built it is transferred via a third core switch, which is part of the so-called BackEnd Network (2.10.1), to one of the Event Filter nodes (2.7) and finally to one of the nodes of the data logging farm (2.8), for local storage and subsequent transfer to the CERN computer centre. Two types of HLT nodes can be distinguished: nodes connected exclusively to the BackEnd Network and nodes connected to both the DataCollection Network and the BackEnd Network, which are referred to as XPUs. These allow additional flexibility as it is possible to move nodes between the L2 and Event Filter farms by adapting the configuration database (2.15.3).

Event processing in the HLT starts with the arrival of RoI information in the RoI Builder (2.4.1). For each event RoI information from the various L1 sources is combined and passed to one of a number L2 supervisor (L2SV) applications (5 in October 2011), running on dedicated processing nodes. Each L2SV schedules events on a unique subset of the L2 nodes.[2] The event is assigned to an L2 Processing Unit application (L2PU) running on one of the nodes and a message with the combined RoI information is sent to that node. The number of L2PUs per node is either equal to the number of processing cores or since 2012 equal to the number of hardware threads ("hyper-threads").

The L2PUs host the event selection software, which requests part of the event data based on the RoI information received. Data request messages are sent to the appropriate ROS PCs, provided the data requested were not already received and stored locally ("cached") as a result of earlier requests. The ROS PCs reply with just the requested data with the granularity of a single ROB.

Each L2PU reports decisions produced to the L2SV from which it received RoI information. Information on how decisions are achieved and which objects are reconstructed etc. is sent to a special type of ROS node. This node is not connected to any front-end electronics, its sole purpose is to store the L2 result information until the Event Builder requests it. The L2 Result Handler application (L2RH) provides the required functionality, three of these applications are running during data taking on a single node.

The L2SVs collect L2 decisions and send them in groups of 100 to the Data Flow Manager application (DFM), which runs on a dedicated node.[3] The DFM assigns each accepted event to an Event Building application (SFI) and sends a message to the SFI assigned. This SFI requests the full event data for the accepted event from all ROS nodes. It uses traffic shaping algorithms to control the timing of the requests to prevent excessive queueing in the network switches. After successful building of an event a message is sent back to the DFM. The identifier of the event is then stored by the DFM in a list of identifiers of events to be deleted. Identifiers of events rejected by the L2SVs are immediately stored in the list. Requests to delete events, each containing a group of

[2] Initially subsets were defined in terms of complete racks of L2 nodes. Improved load balancing has been achieved by allowing different supervisors to manage different nodes in the same rack.

[3] There are 12 nodes for running multiple instances of the application to facilitate running up to 12 independent partitions for testing and calibration purposes.

2016 JINST 11 P06008

typically 100 identifiers are formed using the contents of the list. These requests are sent by means of hardware multicast to the ROS PCs and L2RHs.

After event building the Event Filter Data flow (EFD) component requests the built event which is then transferred to the EFD. At this stage (or strictly once the DFM has caused the event to be deleted from the ROB buffer memories) the EFD has the single remaining copy of the event. It keeps the event in a shared memory virtual disk file, which can be copied to disk if the EFD crashes, thus ensuring that the event data can be recovered even in case of fatal errors. On each EF node there is one EFD application, and multiple Event Filter Processing Unit applications (EFPUs) hosting, like the L2PUs, the event selection software. In this way the data flow is shielded from problems in the EF algorithms, while easy recovery of crashed applications is possible by simply restarting them. The components (EFPUs and EFD) communicate via the shared memory used to store the event data.

Events failing the trigger algorithm selection are dropped unless the trigger is configured to accept those events based on their event type. Accepted events are transferred to the data logging farm, where the SFO applications write the data to disk, in one or more output streams, again depending on the type of event (2.6). Afterwards the data are transferred to mass storage in the computer centre of CERN.

Events accepted by the HLT algorithms are assigned to one of several streams (2.6) depending on which trigger menu item they fired. Events are coarsely classified into physics, express, calibration and debug streams. The physics and express streams are inclusive, so the same event can end up multiple times in different streams. The express stream contains a subset of the events in the physics stream. The debug stream is used exclusively for events where a problem (e.g. crash or timeout) has meant that no decision has been reached. When writing an event to file the SFO considers both stream and luminosity block (1.4) to decide which file or files to write it to and when to close each file. In order not to complicate data analysis it is a requirement that events that belong to the same luminosity block and stream are written to the same set of files.

The common underlying message passing software used for data transfer between applications can use either TCP or UDP network protocols. The message passing mechanism is easily extensible to other protocols. In the case of UDP there is no guaranteed delivery. However, the application level protocols are structured in such a way that exchanges take the form of a request/reply pattern so that errors can be detected by means of time-outs, independently of whether the errors are caused by a network problem, an application crash etc. In practice TCP is required for certain communication paths because the messages are larger than the maximum size of a single UDP datagram (64 kB). Measurements have shown that the performance with TCP is almost equivalent to that achieved with UDP.

None of the DAQ/HLT components can generate a busy signal, unlike the front-end systems. Instead they rely on backpressure between components to temporarily stop the data flow. Explicit XON and XOFF messages are exchanged between various components for this purpose. This allows all available buffer space in the system to naturally fill up until the backpressure (in the form of an XOFF asserted by one of the L2SVs) reaches the RoI Builder, which in turn asserts an XOFF via the links connecting it to the L1 system. It is also possible for a ROB to send an XOFF to the ROD from which it receives data, which may result in the assertion of a busy signal by the ROD. In both cases the L1 system is throttled (L1 accepts are suppressed), leading to dead time.

2016 JINST 11 P06008

2.1.4 High level trigger

The HLT algorithms are mostly developed in an offline software environment and then used inside the online applications (2.9). A plugin architecture allows the online code to load libraries at runtime and to communicate in a well-defined way with them. In addition it allows replacement of the real HLT algorithms with simplified emulation routines that can be used for testing the system.

Several abstract interfaces from the offline environment are re-implemented in the online HLT software in a way that is more appropriate for online running. One example is the Gaudi [31] histogram service, which manages a set of histograms and writes them to a file at the end of a run. Online this is replaced by a version that publishes the histograms to the Online Histogramming Service (2.12.1) so that they can be inspected and analyzed while running.

The execution of the algorithms in the HLT is driven by the trigger menu. This menu determines both which algorithms are to be executed given the decision from the previous trigger level and the exact sequence to be run. In addition prescale values and thresholds are specified in the menu to decide when a given object passes a cut. The HLT Steering part of the HLT software is responsible for coordinating this. It is scheduled by the Steering Controller (2.9.1), a common framework for L2 and EF.

Configuration data that is not related to the menu but to the geometry, or alignment and calibration of the detector are accessed through the geometry and conditions databases, respectively. For online runs typically the most up-to-date conditions approved by the sub-detector experts are used. The detector geometry is loaded at the configuration time of the applications, whereas most of the conditions data are refreshed at the start of each run and there can be multiple runs without reconfiguring the applications. Some conditions data requiring more frequent updates can be reloaded during a run. These include the HLT prescales and the online beam position and size (2.9.2). The large number ($O(10^4)$) of HLT applications that require simultaneous access to the databases require the use of an intermediate proxy and caching mechanism (2.2.4).

2.2 Common software infrastructure

This section describes basic software packages and services used by the TDAQ subsystems: the Inter-Process Communication (IPC) wrapper, the Information Service (IS), and the Error Reporting Service (ERS) and the Message Reporting Service (MRS) and associated archiving of messages, and also the common database infrastructure.

2.2.1 Inter-process communication

In view of the size and the distributed nature of the ATLAS TDAQ system support for inter-process communication by highly scalable distributed middleware with excellent performance is required. Because of the long lifetime of the ATLAS experiment the middleware has to be easily extensible and maintainable. The requirements have been met by adopting the Common Object Request Broker Architecture (CORBA) standard of the Object Management Group (OMG) [32] and making use of the omniORB [33] (for C++) and JacORB [34] (for Java) implementations of the Object Request Broker (ORB).

CORBA has one essential weak point: the complexity of the communication model and of the communication API. This complexity is due to the flexibility offered by CORBA to developers of distributed applications. To overcome this issue, a light-weight software wrapper called IPC has

2016 JINST 11 P06008

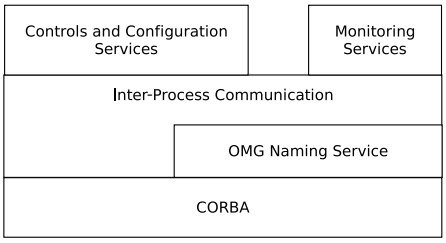


Figure 6. IPC package in the context of the TDAQ software.

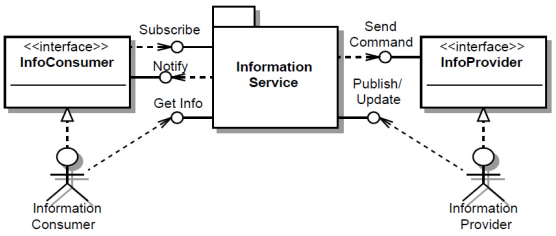


Figure 7. IS interfaces.

been implemented on top of CORBA, as shown in figure 6. The wrapper significantly simplifies the distributed programming interface by narrowing the very wide spectrum of CORBA functions to a reasonably small subset using a simple API and a transparent cache for remote object references. In addition the IPC wrapper provides the notion of a communication domain, which allows multiple instances of the TDAQ online services to be used concurrently and independently of each other. These software communication domains ("IPC partitions") correspond to TDAQ partitions (2.1.2), each containing either one or more TTC Partitions or a service partition. The latter contains only software infrastructure for ad hoc functionality, an example is the mirror partition for remote monitoring (2.12.4).

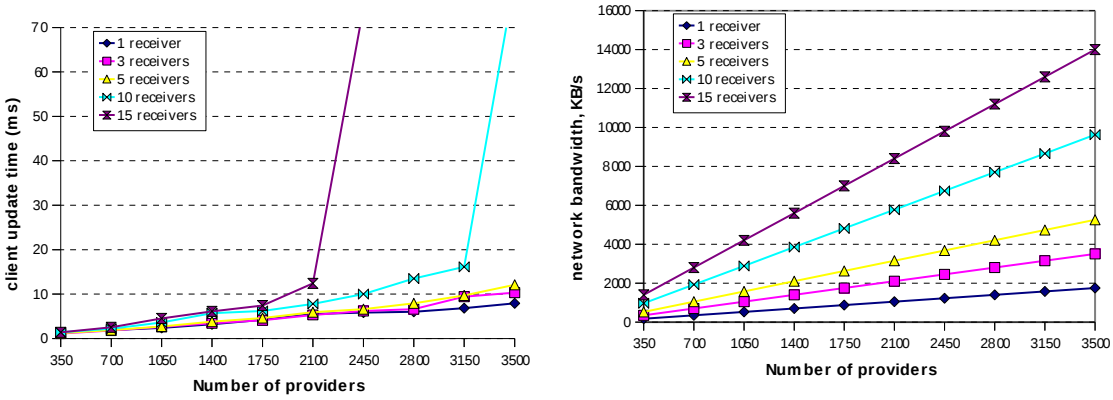
2.2.2 Information Service

The IS provides generic means for sharing user-defined information between distributed TDAQ applications. It implements a client-server communication model, where information is stored in memory by so called IS servers. Any TDAQ application can act as a client to one or several IS servers by using one of the public interfaces provided by the IS, see figure 7:

- an information provider can publish its own information to an IS server using the Publish interface and inform it about changes in the published information via the Update interface,
- an information consumer can either access the information of an IS server on request, using the Get Info interface, or it can receive information updates asynchronously via the Subscribe/Notify interface.

In 2005 the scalability and the performance of the IS have been tested in the context of the TDAQ software large scale tests, organized at CERN [35] with conditions similar to those of real TDAQ running. The behavior of configurations with several thousand information providers and a moderate number of information receivers was studied. Figure 8 shows the results of these tests. The IS server was running on a computer with a dual Pentium IV processor with 2.8 GHz clock frequency and with 2 GB RAM per node. The plot on the left side shows a fast rise of the time to execute one update operation on an IS server as a function of the number of information providers for the case of 10 or 15 information receivers. This is due to the insufficient bandwidth of the Fast Ethernet (100 Mbit/s) network used at the time, the bandwidth required is shown in the right plot.

2016 JINST 11 P06008



(a) Client update time.

(b) Network bandwidth.

Figure 8. IS client mean update time (ms) and required network bandwidth (kB/s) as a function of the number of providers and for several choices of the number of subscribers.

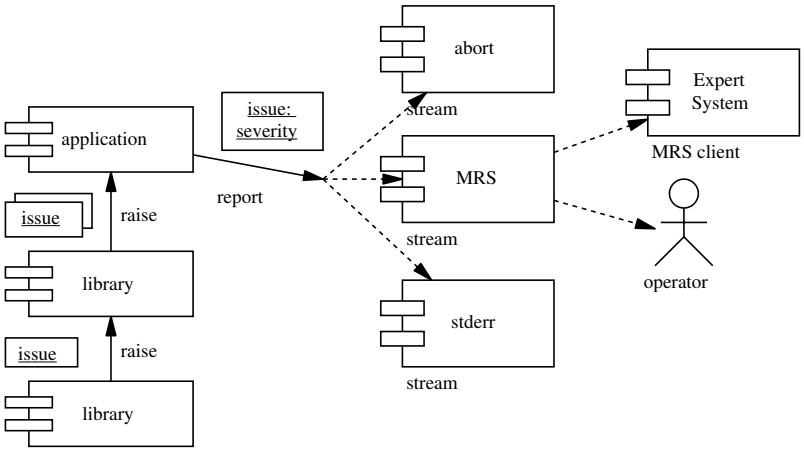


Figure 9. Flow of ERS issues. A library creates an issue as a C++ exception and passes it to a higher level. Finally an application assigns a severity to the issue and reports it to one or more streams.

2.2.3 Error and message reporting and archiving

Every software component of the TDAQ system uses the ERS [36] to report issues (conditions that need attention), either to the software component calling it or to the external environment, e.g. a human operator or an expert system. Issues may be chained when they are passed from low-level libraries to the application level (see figure 9), so that the original cause can be determined from the top-level message. The ERS also provides an interface to report messages to different streams according to their severity. Messages in these streams may simply go to standard output, to the MRS [37], or to specially configured error streams, which may even abort the application in severe cases.

The flow of messages can be seen online by the TDAQ shift operators in the MRS monitor application window (figure 10), which is also integrated in the TDAQ IGUI (2.11.8). The Log Service package [38] implements archiving of the messages for offline retrieval. The package

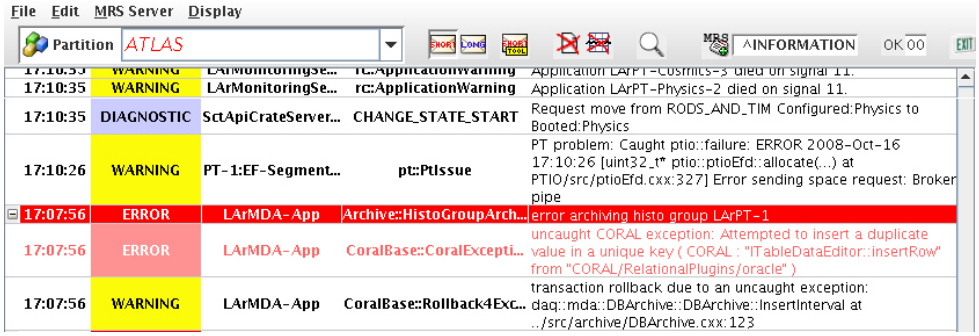


Figure 10. Messages displayed by the MRS monitor application.

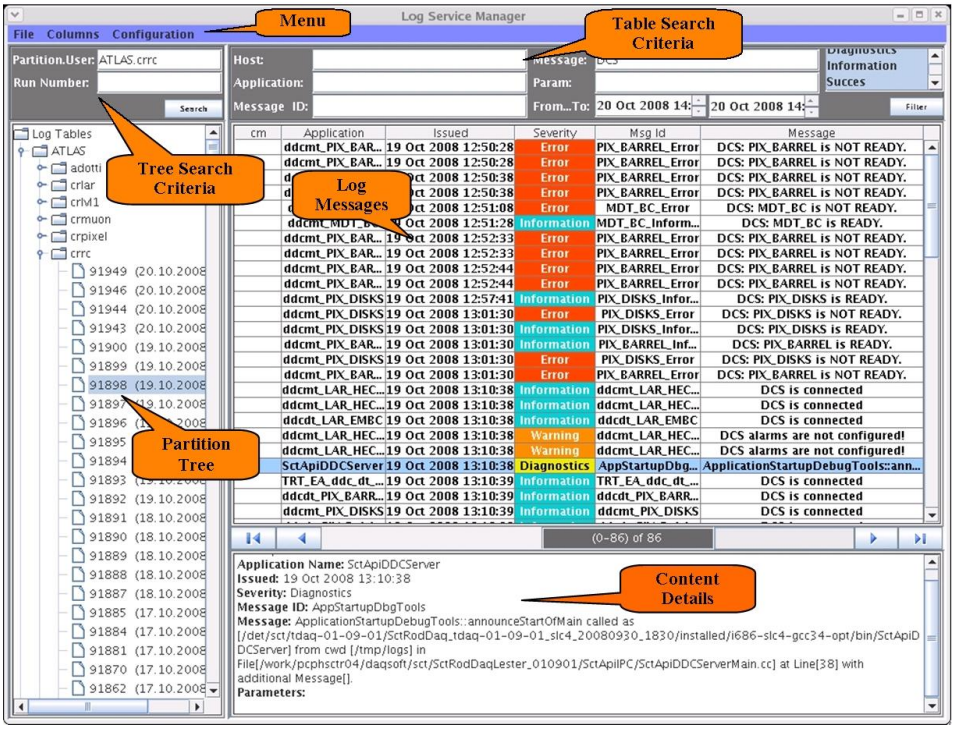


Figure 11. Log Manager GUI window.

provides the Logger application, an MRS client that collects and archives all of the ERS messages flowing in the system in an Oracle database. It also includes a set of command line utilities to access and manage the database and the Log Manager, and a GUI application that provides an intuitive and user-friendly interface to the database for browsing the archived ERS messages offline (figure 11).

Tests have shown that the Log Service can sustain a rate of 4,000 messages per second. This has proven to be sufficient. Even during the frenetic testing and commissioning activity in 2009 with first collisions in the LHC, the peak message rate did not exceed 2,000 messages per second.

2.2.4 Relational database infrastructure

Detector geometry information, trigger configurations and conditions data, as well as selected data from data-taking runs, are stored in relational databases [39]. The main back-end is an Oracle Real

2016 JINST 11 P06008

Application Cluster [40] hosted in the CERN computer centre. The ATLAS online system uses three nodes to serve its needs. Each node has two quad-core CPUs with 16 GB RAM. The total storage capacity is 5 TB spread over 96 disks. To prevent potential bottlenecks, the online database is not directly accessible from the outside but instead is replicated to the ATLAS Tier-0 database on a continuous basis. Similarly, a gateway exists through which conditions updates can be imported that are queued from the offline side.

At the programming level, the relational databases are accessed through a common API called CORAL (COmmon Relational Abstraction Layer) [41, 42], an interface jointly developed by three of the LHC experiments and the CERN IT department that allows technology-independent and SQL-free access from C++ and Python. The CORAL interface frees the application code from any particular database technology. Supported back-ends include direct access to Oracle and MySQL [43] servers as well as local access to SQLite [44] files. This abstraction layer has greatly facilitated the TDAQ commissioning phase during which MySQL servers were used until the final Oracle cluster was deployed, as well as the day-to-day development in which SQLite files are common.

One characteristic challenge of the HLT system is the virtually simultaneous request of (identical) configuration and conditions data from its thousands of processes before the start of a data-taking run. With $O(100)$ MB of data needed by each process, such a load cannot be handled by a single server. To achieve scalability of the configuration and conditions access from the growing number of HLT clients, a dedicated database proxy has been developed for the use-case of the ATLAS HLT that caches the client requests and multiplexes the responses. This so-called CoralProxy uses a custom, technology-independent protocol that essentially implements the CORAL API over the network. On the other side, a multi-threaded server process, the so-called CoralServer, mediates between the proxies and the database back-end [45]. A hierarchy of proxies mirrors the segmentation of the hardware: each HLT node is served by a node-level proxy, each HLT rack is served by a rack-level proxy and each of the L2 and EF farms is served by a top-level proxy. Thus the database server sees only a single client, while each HLT client talks to a local database server. This has been demonstrated to achieve full scalability. Another advantage of the CoralServer/CoralProxy infrastructure is that it handles the authentication of the database clients by deferring it to the CoralServer. Thus, the HLT clients no longer need to store credentials for access to the Oracle database.

2.3 Readout system

2.3.1 System overview

The ReadOut System (ROS) receives and buffers event fragments from the RODs upon L1 accepts and forwards them on request to the L2 system or to the Event Builder. The event data are input via the ROLs, which cross the boundary between sub-detector specific readout electronics and the DAQ system.

The ROS is built from 151 rack mountable, 4U high, PCs. The number of ROS PCs and the number of ROLs for each sub-detector are specified in table 1. The ROLs are connecting to purpose-built PCI cards, the ROBIN cards, residing in the PCs. Most PCs contain four ROBIN cards. One ROBIN card has three ROL inputs and for each ROL a ROB (ReadOut Buffer). Each PC is connected to the DataCollection Network and to the Control Network, see figure 5.

2016 JINST 11 P06008

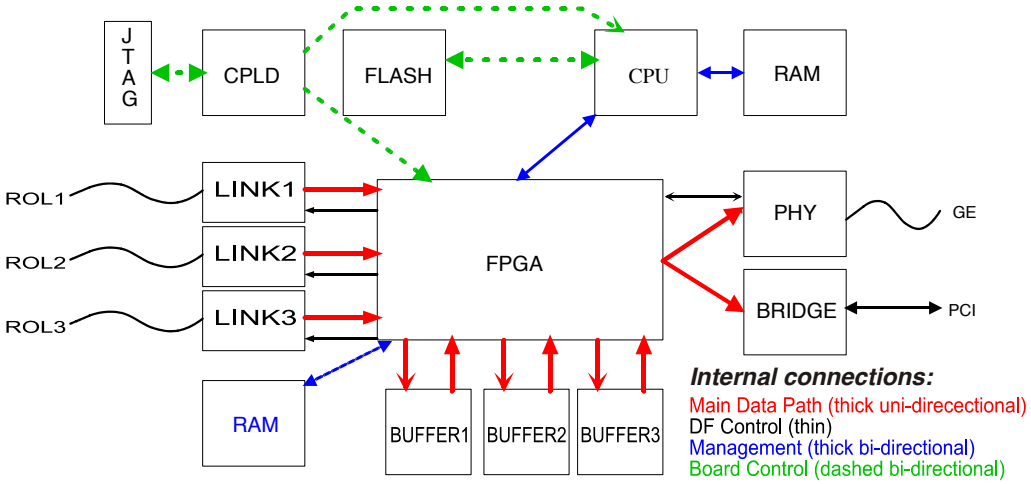


Figure 12. Block scheme of the ROBIN.

Figure 13. Photograph of the ROBIN.

2.3.2 The readout link

The ROL is implemented as a dual optical fiber link running the S-LINK protocol [18] with either 160 or 200 MB/s net throughput. The protocol supports the use of control words that can be distinguished from event data. This is possible due to the 8b/10b coding [46] used on the link. Each event fragment is preceded by a "Beginning Of Fragment" (BOF) control word and followed by an "End Of Fragment" (EOF) control word. For each event fragment a Cyclic Redundancy Checksum (CRC) is generated by the interface to the link of the ROD and checked by the ROBIN, allowing detection and signaling of bit transmission errors. The S-LINK protocol employs XON-XOFF signaling to prevent buffer overflow. Assertion of XOFF by the ROB causes the ROD to stop outputting data, which may cause it to raise its BUSY signal and halt the L1 trigger.

2.3.3 The ROBIN

The ROBIN [47] is a plugin card for 64-bit, 66 MHz PCI slots. A block scheme of the ROBIN is shown in figure 12, a photograph in figure 13. The ROBIN implements three ReadOut Buffers, each with 64 MB of memory. The buffers are dual-ported, each port of each buffer can sustain a data transfer rate of more than 200 MB/s, the maximum bandwidth of the ROL. The buffers are paged, the page size is programmable (from 1 to 128 kB) and has a typical value of 2 kB. The three buffers are managed by a Xilinx Virtex-II 2000 FPGA [48] and an on-board PPC440GP PowerPC processor [49] running at 466 MHz, which has 128 MB of main memory. A FLASH memory of 8 MB stores executable code for the processor, the bit stream for configuring the FPGA, some data needed for configuring the software running on the processor and, in a one-time-programmable sector, a serial number and manufacturing information. A Complex Logic Device (CPLD) takes care of resets and of JTAG interfacing. A dedicated bridge (PLX PCI 9656 [50]) is used for interfacing to the 64-bit PCI bus. The ROBIN has a Gigabit Ethernet (GbE) interface, intended for providing additional output bandwidth. It is implemented in the FPGA and has a dedicated transceiver (PHY). However, it was found that the benefit of using the interface is marginal, because of the processing power required to serve the port. Furthermore it was also found that upgrading of the motherboard, CPU and memory of the ROS PC, as described in 3.1.2, results in a substantial increase of the maximum throughput of the ROS PC. The GbE interfaces of the ROBINs are not used in view of this and also in view of the impact on the DAQ software. Each board also has a connector for 100 Mbit/s Ethernet connected to the Ethernet port of the PowerPC processor, which can be used for management purposes. An RS-232 connection is also available and can be used for communicating with a simple monitor program (U-Boot [51]). By means of a dedicated driver an RS-232 interface is emulated that can be accessed via the PCI interface. The emulated interface allows communication with the monitor program without a physical connection between a suitable serial interface (typically the interface of the PC) and that of the ROBIN.

Figure 14 illustrates how the event data are handled by the ROBIN. Event data flow from the ROLs into the buffer memories. For test purposes data can alternatively be generated by data generators or input from FIFOs. The latter can be filled with arbitrary data patterns by the processor. For each event fragment received or generated a Cyclic Redundancy Check (CRC) checksum is formed while the fragment is passed to the buffer memory. Data are stored in free pages of the buffer memories and are retrieved from the buffer memories by the Direct Memory Access (DMA) engine. Identifiers of free pages are provided to the buffer managers via the Free Page FIFOs. The buffer managers exert backpressure if these FIFOs are empty. For normal data taking the backpressure halts the data flow and results in XOFF signals on the ROLs (each ROL handler contains a 256 word FIFO to prevent data loss), otherwise either the data generators are stopped or data are no longer input from the test input FIFOs. The processor supplies identifiers of free pages to the Free Page FIFOs (with a size of 1024 words each) and receives for each used page four words (containing status and error information in the first word, the L1Id in the second word, page number and length of data stored in the page in the third word, the last word is reserved for the run number but is not used) via the Used Page FIFOs. Each Used Page FIFO can store 256 blocks of 4 words. The processor keeps track of the data stored in the buffer memories on the basis of the information received via the Used Page FIFOs. It also retrieves commands written via PCI bus to the Dual Port Memory

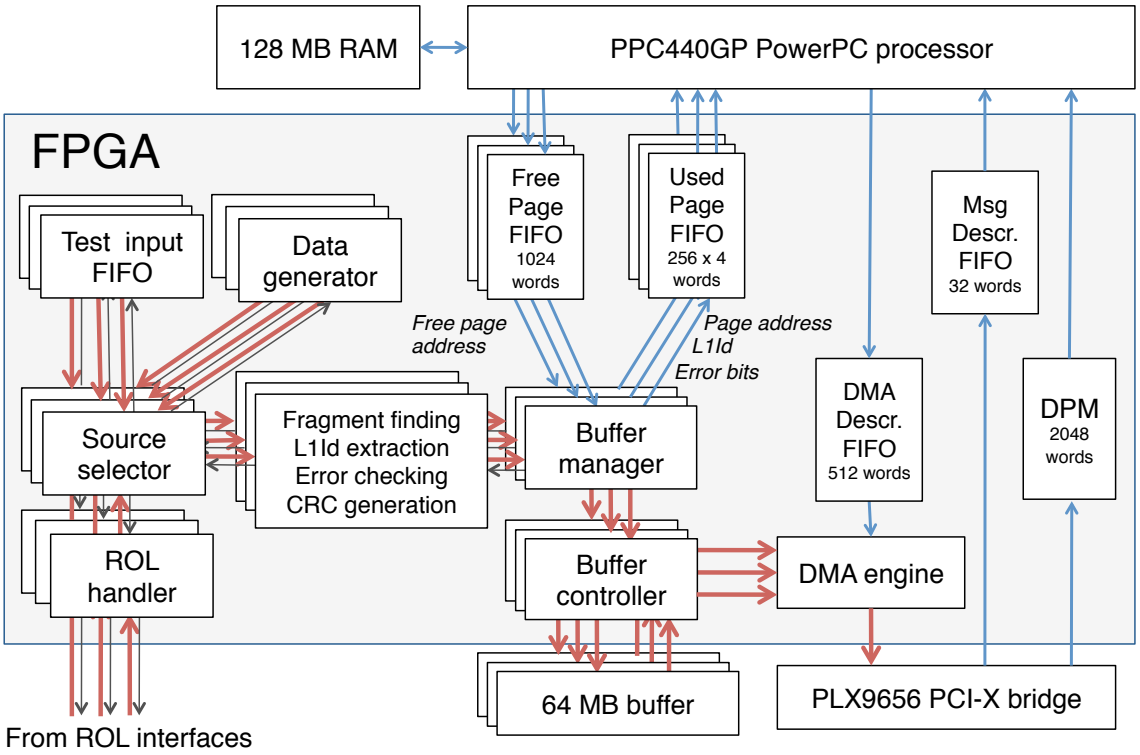


Figure 14. Block scheme of the configuration of the FPGA.

(DPM) and descriptors associated with the commands written to the Message Descriptor FIFO and handles the commands. For each message one word is written to this FIFO indicating the nature of the command stored in the DPM. In case of a request for data the processor forwards information on the location of the data in the buffer memories to the DMA descriptor FIFO as well as an header, the ROB header, for the event fragment. The header is first stored by the DMA engine in the memory of the PC, then the event data are appended and finally a CRC computed by the ROBIN may be added, depending on how the ROBIN is configured. Responses to messages other than request messages are written to the DMA descriptor FIFO and are also transferred under DMA control to the memory of the ROS PC. In the current implementation of the ROBIN software event fragments can only be requested per ROB, therefore three separate requests have to be forwarded to the ROBIN if the three fragments of the same event stored in the three buffer memories have to be transferred to the memory of the ROS PC. Delete requests to the ROBIN have also to be provided individually per ROB, but in one request it is possible to specify up to 100 events to be deleted by providing their L1Ids. The processor handles these requests by writing identifiers of pages to be freed to the Free Page FIFO.

The program running on the processor of the ROBIN has been written in C and consists of a loop in which data stored in the Used Page FIFOs and the Message Descriptor FIFO is read and handled, and in which identifiers of free pages are written to the appropriate Free Page FIFOs. The relative service rates, as well as other parameters, such as the page size of the buffer memories or the temperature above which an alarm will be caused, are configurable and are stored in "environment variables". The contents of these are stored in the FLASH memory of the ROBIN and can be set either by sending appropriate commands to the ROBIN or with the U-Boot monitor. As the standard

2016 JINST 11 P06008

values of the "environment variables" as well as the software for the PowerPC processor are stored in the FLASH memory there is no need to boot the ROBINs from the ROS PC after powering it up. The ReadoutApplication, the program running on the ROS PC for handling requests for event fragments and forwarding the data requested, can also send configuration information to the ROBIN on the basis of information specified in the configuration database. The ROBIN software keeps track of e.g. the number of event fragments received, the number of requests received for which the fragment requested could be provided and the number of requests for which this was not possible, etc. This information, together with the version numbers of the software and of the firmware and configuration information, is passed upon request to the ROS PC. A dedicated program, "robinscope", can request and display the data for debugging purposes.

The ROBIN firmware and software check for error conditions. Errors detected are signaled in the ROB header in a status word, see table 3. It is possible to configure whether or not errors give rise to PCI interrupt requests. Corrupted event fragments that cannot be requested in the normal way (e.g. because the L1Id is missing) are stored in a reserved part of the buffer memory and can be retrieved with the help of special commands, passed via the Message Descriptor FIFO and the DPM as described above.

2.3.4 The ROS PC

Until the summer of 2011 all ROS PCs were equipped with a SuperMicro X6DHE-XB motherboard [52] with six 64-bit PCI-X slots and one 4-lane PCIe slot, one 3.4 GHz Intel Xeon processor (single core, Irwindale [53]) and 512 MB of memory. Since then the motherboards, CPUs and memory of 107 PCs have been gradually replaced by Supermicro X7SBE motherboards [54] with four 64-bit PCI-X slots and two PCIe slots, 3.0 GHz quad-core CPUs (Intel Core 2 Q9650 [55]) and 4 GB memory, respectively. The configuration of most ROS PCs is as schematically shown in figure 15: 4 ROBINs are placed in 4 PCI slots, associated with either 4 or 2 PCI-X segments for the X6DHE-XB and X7SBE motherboard respectively. The PC connects to the DataCollection Network by means of two ports of a PCIe GbE interface (X6DHE-XB: 4-lanes, 4 ports, Silicom PEG-4 [56], X7SBE: 2-lanes, 2 ports, Silicom PEG-2i [57]. One of the network ports of the motherboard is connected to the Control Network. Each PC has a triple redundant power supply and an IPMI interface [58], allowing remote control (power off, power up, reset) and monitoring (temperatures, fan speeds) of the PC via the Control Network. The operating system of the PC is Linux (SLC5 [59]), the PCs are netbooted (again via the Control Network) and do not have disks.

A multi-threaded application, the ReadOutApplication, forwards requests received via the DataCollection Network to the ROBINs, and sends event fragments received from the ROBINs via the network to the L2PUs and SFIs requesting the data. It also forwards delete requests, received from the DFM, to the ROBINs. Each request is dealt with by a separate thread: the Request Handler. Upon receipt requests are stored in a queue and assigned one by one to these threads, i.e. a single Request Handler deals only with one request. The maximum number of Request Handlers is configurable, a typical number is 12. Each Request Handler communicates with the ROBINs and requests data from the individual ROLs as needed. If available, these data are transferred (under DMA control) to the memory of the PC, otherwise an empty fragment with error bits set is passed to the PC. The memory in question has contiguous physical addresses and is allocated once by a special driver: the cmem driver [60]. Via this driver the memory can be accessed. The event fragments written by

2016 JINST 11 P06008

Table 3. Error conditions signaled in the ROB header. Bits 0–5 are general purpose error bits, also used in other types of headers, bits 16–31 are ROBIN specific. BOF and EOF refer to the control words passed via the ROLs indicating event boundaries (2.3.2).

Bit	Description
31	Discard: the ROBIN did not have a fragment for the requested L1Id because it is in discard mode. It therefore generated an empty fragment.
30	Pending: the ROBIN did not have a fragment for the requested L1Id but this fragment may still arrive. It therefore generated an empty fragment.
29	Lost: the ROBIN did not have a fragment for the requested L1Id. It therefore generated an empty fragment.
28	Short fragment: the amount of data between the S-Link control words (BOF and EOF) was less than the size of an empty ROD fragment (ROD header + ROD trailer).
27	Truncation: the amount of data sent across S-Link for this fragment was larger than the maximum fragment size the ROBIN was configured to handle. Therefore this fragment has been truncated.
26	Tx error: general flag for an S-Link transmission or formatting error. See bits 17 thru 23.
25	Sequence error: the L1Id of this ROD fragment was not in sequence with the L1Id of the fragment previously received (L1Id_new not_equal L1Id_old + 1).
24	Duplicate event: when this fragment was received the ROBIN still had a fragment with the same L1Id in memory. The new fragment has replaced the older one.
23	Double BOF: two successive BOF control words received.
22	Double EOF: two successive EOF control words received.
21	Missing BOF: new fragment started without BOF (after preceding one terminated with EOF).
20	Missing EOF: new fragment started with BOF, without the preceding one terminated by EOF.
19	Incomplete header: number of header words between BOF and EOF lower than threshold.
18	No header: EOF word immediately followed BOF word.
17	CTL word error: S-LINK transmission error on control word (EOF or BOF).
16	Data block error: S-LINK transmission error on data block.
5	An overflow in one of the internal buffers has occurred. The fragment may be incomplete.
4	Data may be incorrect, further details provided by bits 16–31.
3	A time out has occurred, the fragment may be incomplete.
2	An internal check of the L1Id has failed.
1	An internal check of the BCId has failed.
0	Unclassified.

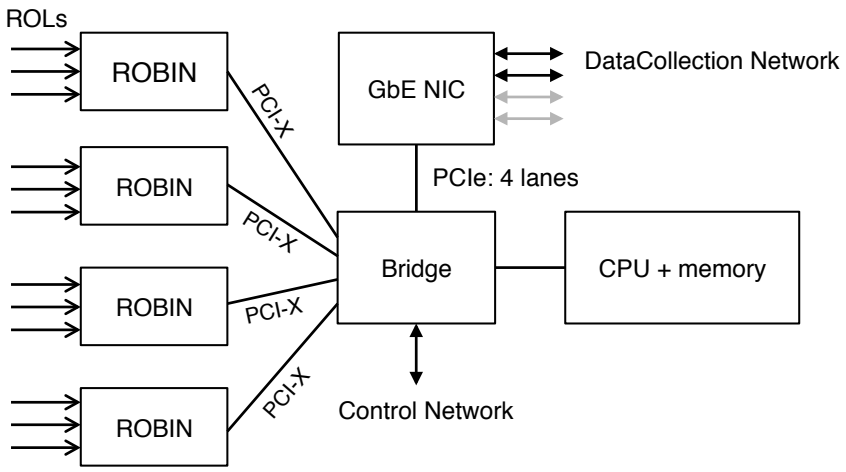


Figure 15. Configuration of a ROS PC equipped with the SuperMicro X6DHE-XB motherboard.

the ROBIN consist for each ROL of a ROB header followed by a ROD fragment and optionally by a CRC generated by the ROBIN. The ReadOutApplication will request the data again from the ROBIN after a configurable timeout if an empty fragment was received (this can typically occur in a test situation where requests may arrive before the fragments requested arrive). The fragments received from the ROBs, once all have arrived in the memory, are concatenated and sent to the requester.

Three different types of requests can be distinguished: Event Builder (EB), L2 and L2 E_T^{miss} requests. EB and L2 E_T^{miss} requests are forwarded to all ROBs in the ROS PC, L2 requests only to the ROBs specified in the request. L2 E_T^{miss} requests are only sent to ROS PCs receiving calorimeter data. These data contain sums of energy deposits calculated in the calorimeter RODs. Only these energy sums, 6 words for each ROB, are passed to the L2PU sending an L2 E_T^{miss} request and are used for the second-level missing energy trigger. This trigger is in use since early 2012 and runs at a rate of about 10 kHz. The upgrade of the ROS PCs, in combination with the introduction of the L2 E_T^{miss} requests, made this trigger feasible. In principle normal L2 requests could have been used, requesting data from all ROBs, however, the bandwidth provided by the two GbE links would then not be sufficient the transfer of all of the data to be transmitted out of the ROS.

Errors detected by the ReadoutApplication are signaled in a header that is prepended to the response message. This header is removed by the L2PU or SFI receiving the message. However, error information found in message headers of this type is propagated by the SFIs to the event status information in the event headers constructed by the SFIs. The latter type of headers as well as the ROB headers are part of the event data stored for offline analysis, so that the error information is available offline. If an error is that of an empty fragment the data may be requested again by the L2PU or SFI.

To allow use of the ReadOutApplication for testing, with different hardware than ROBINs, and for applications requiring functionality provided by it, hardware or application dependent parts have been implemented as dynamically loadable libraries (plugins). The plugin for communicating with ROBINs may for example be replaced by one for handling event data arriving via alternative inputs, for instance via Ethernet. It is also possible to use a plugin for preloading event data in the ROS PC for DAQ system tests (2.16). For small scale testing the plugin to handle requests arriving via the network can be replaced by a plugin autonomously generating requests for the ROBINs and the output of the ReadoutApplication can be transferred to a local disk or a disk accessible via the network. The plugins to be loaded are specified in the configuration database.

2.3.5 ROD Crate DAQ

The ReadOutApplication is also deployed, with appropriate plugins, as ROD Crate DAQ (RCD) application [61]. Most RODs are VME modules and are installed in VME crates equipped with a single board computer with a VME interface and running SLC5 [59]. The RCD application runs on the single board computer, together with the standard DAQ software infrastructure. Its main tasks are control and collection of event data from the RODs for monitoring, calibration and testing purposes. These tasks are similar to those of the ROS PC, although the performance requirements are considerably less. Communication with the different types of RODs via the VME bus is achieved by means of ROD specific plugins.

2016 JINST 11 P06008

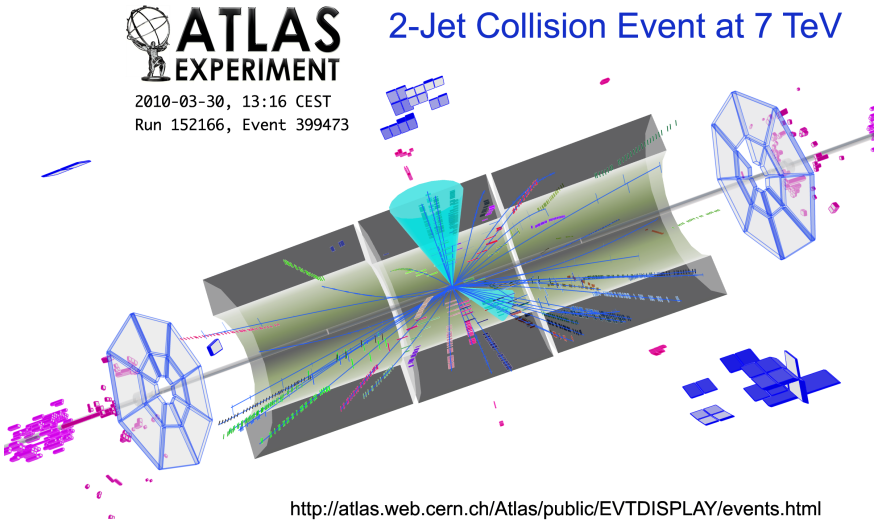


Figure 16. Event display showing production of two jets.

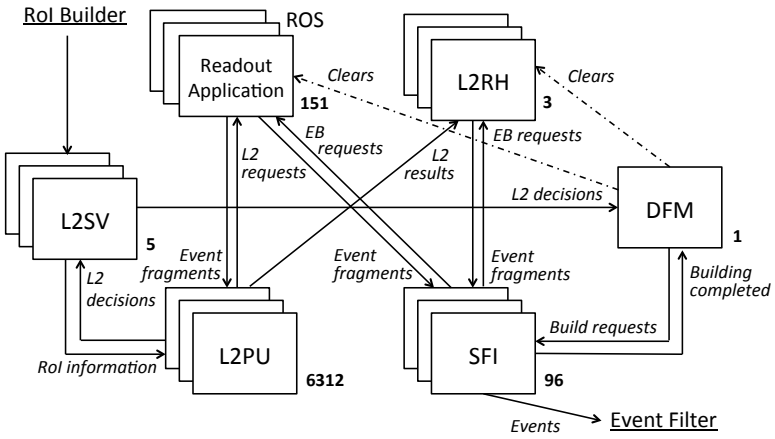


Figure 17. Message exchange between the L2 and Event Builder applications. Event building is initiated by the DFM for events accepted by L2. The number of instances as deployed in October 2011 is indicated.

2.4 L2 system

As described in 2.1.1 the L2 trigger is guided by Region of Interest (RoI) information, produced for each event accepted by the L1 trigger and based on the energy deposits in the calorimeters and muon track segments found (1.3). The L2 trigger uses this information for fetching a subset of the event data from the ROS. The event display depicted in figure 16 illustrates this: based on the energy deposits of two jets in the calorimeter the L1 trigger will have identified two RoIs and will have provided the approximate co-ordinates, in the form of η/ϕ indices, of their centres to the L2 system. This causes the L2 trigger to request calorimeter data originating from areas around these locations.

The L1 trigger recognizes 4 different types of RoIs [3, 21, 24]: muon, electron/photon (also referred to as electromagnetic), τ and jet RoIs. The last three types are all based on energy

2016 JINST 11 P06008

deposits in the calorimeters, but the deposits have different characteristics: electron/photon RoIs are relatively narrow and contained in the electromagnetic calorimeters, τ RoIs are less narrow than electron/photon RoIs and typically extend into the hadronic calorimeters, while jet RoIs are much wider and also typically extend into the hadronic calorimeters. Which data is requested from the ROS and how the data is processed by the L2 trigger depends therefore on the RoI type, but not exclusively. For example, for validating an electron/photon RoI a small part of the calorimeter data needs to be requested from the ROS. If the RoI is likely to have been caused by an electron, a matching track will be searched for in the inner detector. For this, data from a small part of the pixel detector, the SCT and the TRT needs to be fetched from the ROS (the spread in the positions of the collisions along the beam direction causes the regions from which the data requested originated to be elongated in the η direction). More details on the usage of RoIs can be found in [29].

The L2 trigger system consists of the Region of Interest Builder (RoIB), a small number of nodes running the L2 Supervisor application (L2SV), a large farm of nodes running the L2 Processing Unit application (L2PU), and a few nodes running an instance of the L2 Result Handler application (L2RH), as described in 2.1.3. Figure 17 shows an overview of the messages exchanged between the L2 applications, the ROS and the Event Builder. In the next sections L2 specific hardware (the RoI Builder) and software are described. This is followed by an overview of how fault tolerance is achieved in the L2 system and by a description of support provided by it for calibration of the muon precision chambers.

2.4.1 The RoI Builder

Table 4 contains an overview of the information output by the L1 trigger for each accepted event. Together with positions of the four different possible types of RoIs, information on the trigger decision is passed via 8 S-link [17, 18] connections to the RoI Builder (RoIB) [62]. For each event the RoIB collects and merges the information into a single message that is forwarded to one of the L2 Supervisors, again via an S-link connection.

The RoIB is a VME based system that includes a controller which configures and monitors the system along with custom cards that assemble the event fragments and distribute them to the L2 Supervisors, see figure 18 for a block scheme and an overview of the links into and out of it. There are two card types. The input card accepts three inputs from the L1 subsystems or the TTC [7]. The builder card assembles the input data of a subset of the events and passes the results to one of four L2 Supervisor S-link outputs. The fragments are identified by a 32 bit id, the extended L1 Id (L1Id), described in 1.5. The TTC input to the RoIB formulates an L1Id by counting L1 accepts and ECRs as they arrive. The L1Id word appears in the same location in each of the serial input streams. The input cards use this word, modulo a configurable value, (typically the number of outputs enabled) to key the various fragments and send them to the builder card and output channel that has been assigned this key value.

The input data is transferred over a custom J3 backplane. The backplane operates at 20 MHz and transfers 16 data bits per clock cycle simultaneously for up to 12 inputs. The total maximum data throughput is therefore 480 MB/s, 40 MB/s per input. For 100 kHz maximum L1 accept rate the sizes of the fragments input from L1 should therefore be smaller than 100 32-bit words on average. The maximum size of any single fragment is limited to 128 words. This limit is imposed by

2016 JINST 11 P06008

Table 4. L1 data and link count for inputs to the RoIB.

Level 1 system	S-links	Data transferred
Central Trigger Processor	1	Input state (160 bits), trigger decision (256 bits), trigger type (8 bits),GPS time (32 bits), internal trigger data (32 bits).
Muon System	1	List of up to 16 p_T ordered muon candidates, including threshold passed and location of the candidate (32 bits each).
Calorimeter $(e/\gamma, \tau)$	4	Thresholds passed (16 bits), saturation flag (1 bit), position data (12 bits) for each trigger entity.
Calorimeter (jet and energy sum)	2	Thresholds passed (12 bits), saturation flag (1 bit), position data (10 bits) for each jet, E_x (16 bits), E_y (16 bits), ΣE_T thresholds passed (4 bits), ΣE_T (16 bits), missing E_T thresholds passed (8 bits), jet E_T sum thresholds passed (4bits).

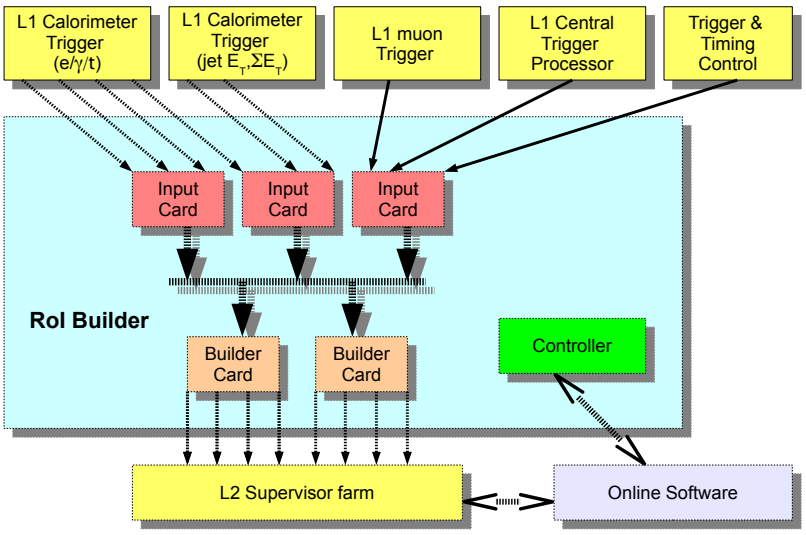


Figure 18. Block scheme of the RoI Builder and overview of connections to external systems. The custom input and builder cards and the controller, a commercially available single board computer, are installed in a single 9U VME crate. The controller connects to the Control Network.

2016 JINST 11 P06008

resources available in the FPGA implementation. These size limitations are not serious constraints since the typical fragment size is 18 words.

The crate can be equipped with up to four of each type of card. This accommodates 12 inputs and 16 outputs. The three input cards and two output cards used cover the needed eight inputs from the L1 system at the full 100 kHz rate and the input of the TTC information.

The input cards are 9U with four Altera APEX 20K200E FPGAs [63]. The cards are driven by a common 20 MHz clock, provided via the J2 backplane by a dedicated clock card. For the inputs they use three mezzanine cards. S-link inputs use the HOLA (High Speed Optical Link for ATLAS) mezzanines designed by CERN [64]. For the TTC input a pin for pin compatible mezzanine was developed explicitly for the RoIB. The builder cards are 9U with five Altera APEX 20K200E FPGAs and they use four CERN HOLA mezzanine cards for the S-link output.

The system was designed to allow for reasonable flexibility with regard to the input and output channels used. This is particularly valuable if there is a need to reconfigure which L2 Supervisors are active. The primary limitation on the number of outputs needed is set by the rate at which the L2 Supervisors can service the inputs from the RoIB. The software and hardware set this limit at four L2 Supervisors to support a full 100 kHz L1 accept rate.

The S-link flow control (XOFF) mechanism is used to throttle the L1 system when the downstream or RoIB system can not support the incoming rate. For this reason the RoIB will relay an XOFF to the L1 system when it receives a fragment that can not be accepted because the builder's output FIFO is full or bus transfers are blocked. This can occur if any of the L2 Supervisors asserts an XOFF and ceases to accept fragments. To prevent a deadlock when an incoming fragment is lost a timeout mechanism was incorporated to allow for partial assembly in cases where one or more L1 subsystems fail to provide inputs within a programmable time period.

A number of test and diagnostic facilities were built into the RoIB. Memory on each of the input cards can be used in two modes. It can be set through the control computer and used to simulate inputs at programmable rates for testing. During normal running it can keep statistics on fragment assembly times and errors which can be interrogated by the control computer and reported to the run control system.

Extensive testing was done, first in coordination with individual L1 components and then with the full system in place. The RoIB was used well in advance of delivery of beam and was tested using real inputs during cosmic ray runs as well as with a computer system equipped with S-link source cards[4] to emulate inputs from L1.

2.4.2 The L2 Supervisor

Each L2 Supervisor node is equipped with a FILAR card [65] used to receive RoI input from the RoIB via S-link. The L2 Supervisor application (L2SV) is implemented as a multi-threaded application of which an instance runs on every L2 Supervisor node. The L2SVs assign each event accepted by L1 to an L2PU application. RoI and other information from the L1 trigger received from the RoIB is forwarded to the selected L2PU. Once information on the L2 decision produced by the L2PU has arrived, the assignment algorithm is informed that L2 processing of the event has finished and the information received is communicated to the DFM.

[4]These are FILAR S-link destination cards [65] with modified firmware, functioning as S-link source cards.

A unique subset of the L2PUs is assigned to each L2SV. At configuration time each L2SV builds a list of these L2PUs. This list contains several entries for each L2PU, as many as the number of events that each L2PU is allowed to queue. This value is typically set to two. Furthermore the list is initially ordered such that successive entries are for L2PUs running on different nodes. The list is used to assign events to the L2PUs: for each event the RoI information received from the RoI Builder is sent to the first L2PU in the list, after which the entry is removed. When the L2SV receives the L2 decision information, the identifier of the L2PU that sent it is added to the end of the list, making a new processing slot available. Initializing and handling the list as described ensures that the available processor resources are used in an optimal way.

The L2SV plays an important role in error handling (2.4.5), and in flow control: the L2SV applies backpressure to the RoIB by asserting XOFF on the connecting S-link if there are no L2PUs available for trigger processing. Likewise, the DFM may send backpressure messages to the L2SV in case the event building cannot proceed. This may happen if the EF or the data recording cannot keep up with the rate. Four nodes are sufficient to reach the maximum design L1 accept rate of 100 kHz as a node running a single L2SV is able to sustain a rate of 25 kHz, but for better fault tolerance 5 nodes are used.

For testing the DAQ/HLT data flow and the selection software it is possible to preload RoI data from files, either Monte Carlo or earlier recorded data. Likewise, the corresponding ROB data may be preloaded in the ROS PCs. It is then possible to run the entire system as during normal data taking. Instead of preloading RoI data into the L2SVs it may be preloaded into the test memory of the RoIB for short runs. Finally, the L2SV may be configured to generate dummy RoI information which is useful for tests of the data flow only.

2.4.3 The L2PU

The L2PU is an application hosting the L2 event selection software. Typically one instance is running per CPU core or per hyper-thread; in October 2011 the total number of instances in the L2 farm was 6312. Each L2PU processes one event at a time. The L2PU is multi-threaded for most of its aspects except for the selection software which runs in a single thread.

The L2PU obeys the standard run control commands. Trigger configuration, conditions and calibration data are read from a database using a proxy to avoid overloading the server (2.2.4). Most of these parameters remain constant for the duration of a run, an exception are the trigger menu prescale values that can be changed dynamically while running, typically when the luminosity becomes smaller than a certain threshold. The prescale factors may vary between zero and one, making it possible to disable or enable complete trigger chains. More details are given in 2.9.2. L2PUs may be stopped and restarted individually to recover from serious error conditions during a run.

For each event assigned to it the L2PU receives the L1 Result data from the L2SV with which it is associated. Then it initiates the execution of the selection algorithms for this event. During this process, the L2PU requests detector data from ROS PCs. At the end of the selection procedure information on the decision is returned to the L2SV, and for each accepted event data generated by the L2 event selection algorithms are sent to an L2RH.

RoI data is retrieved from the ROS PCs as needed, caching of the data retrieved prevents multiple transfers of the same data, see 3.5 for examples of the effect of this. The minimum data

granularity is a ROB; requests usually span several ROS PCs. The amount of data requested ranges typically from a few to 50 kB per event (3.5).

Histograms, produced by the L2PU itself and by the event selection software (communicated to the L2PU via the interface to the event selection software (2.9.1)), are used for monitoring and rate metering and are published periodically (about every minute). This results in about 100 000 histograms which are collected and summed by a tree structured system of gatherers as described in 2.12.1.

In the Technical Design Report [30] it was specified that the L2 processor farm should have the processing power of 1000 single core 8 GHz CPUs (with 2 CPUs per node), a time budget of 10 ms per event (so that the maximum L1 accept rate is 100 kHz) and a data volume of 3.2 MB/s per CPU. However, the initial batch of nodes installed consisted of dual 4 core 1.86 GHz CPU machines, with 30 nodes in each of 27 XPU racks. A typical number used for L2 processing at the time was 10 racks, resulting in an L2 farm with 300 nodes and in total 2400 cores and therefore running 2400 L2PUs, a factor 2.4 larger than originally foreseen. With the 6312 L2PUs in use in October 2011 this factor had grown to about 6.3.

2.4.4 The L2 Result Handler

The L2 Result Handler (L2RH) is an application that runs on a standard server PC[5]. Its function is to buffer the L2 Result and pass it upon request to the Event Builder. The L2 Result contains the L1 Result and details of the L2 decision process in the form of a ROD fragment. Fragments are deleted from the buffer following delete commands sent by the DFM to all ROS PCs and to the L2RHs. Initially, one L2RH was used. However, the increasing number of L2PU applications (growing with the number of cores per machine) together with a larger size of the L2 Result as a result of more complex trigger menus (typically about 20 kB but larger if debugging is enabled) caused contention. It therefore became necessary to run with multiple (configurable, normally three) L2RHs.

2.4.5 L2 fault tolerance and error handling

The co-operating components of the L2 subsystem rely on correct communication. In the absence of error detection and correction, failures may lead to a serious degradation of data throughput or even complete blocking of the data flow. Timeouts are an important means of detecting non responsive applications as well as communication errors:

- A non responsive ROS PC causes a timeout, which results in the production of a dummy ROB fragment by the Data Collection thread of the L2PU. The selection software decides whether it can continue with the event processing or not, and in the latter case the event is classified as a debug event (2.6.1).
- Malfunctioning of the selection software may result in a timeout: a separate thread sets a warning when the timeout approaches. This may be sensed by the selection software and allows graceful termination of the processing. The timeout thread will produce a dummy L2 Result with the status bits indicating an error, classify the event as a debug event and abort the L2PU if the selection software becomes unresponsive. An aborted or crashed L2PU is detected by the control software, which may restart the application.

[5]The node on which the L2RH runs has in the past also been referred to as pROS or pseudo ROS.

- L2PUs detect malfunctioning of an L2RH by a timeout when sending an L2 Result. They will automatically switch over to another L2RH if necessary. Run Control may put recovered L2RHs back into the active system.
- The L2SV will flag unresponsive L2PUs, i.e. L2PUs for which an L2 Decision did not arrive within a certain time interval, as unavailable. Events concerned are marked as unprocessed and as debug events. The L2 Result will be missing in this case. Possible unprocessed events in the L2PU input queue are re-assigned to other L2PUs. The run control is informed and may attempt to restart the affected L2PU.
- In case of failure of an L2SV the RoIB should detect a busy signal (an XOFF generated by the FILAR card in the L2 Supervisor node) on the connecting S-link and stop forwarding L1 Decisions to this L2SV. This will eventually cause the L2PU sub-farm associated with the L2SV to become idle, but the RoIB continues outputting L1 Decisions to the remaining L2SVs.

2.4.6 Support for calibration of the muon precision chambers

For the calibration of the MDT (Monitored Drift Tubes) sub-detector, data corresponding to muon track candidates need to be recorded with a rate of about 1 kHz, to make it possible to calibrate the full MDT system once per day. For analyzing the data three remote analysis centres have been set up. The muon track candidates are found by the L2PUs as a product of normal trigger processing. For each L2PU there is a circular memory buffer for storing the data of selected track candidates (typically a p_T larger than 4 GeV is required). A track candidate is skipped if its data cannot be stored because the buffer is full. On each L2 node one application collects the data stored in the buffers associated with the L2PU applications. In each rack these data are forwarded to an application running on the dedicated node set aside for general per-rack infrastructure services. These applications send the data to mass storage, from where the data is passed to the three calibration centres. For further information see refs. [66, 67].

2.5 Event Builder

The task of the Event Builder [68] is to assemble and format the data of events accepted by L2 and to make the complete events available to the next filtering level, the Event Filter (2.7). The SFIs (Sub-Farm Input applications) perfom the actual event building process. The DFM (Data Flow Manager application) is the supervisor of the Event Builder farm consisting of the nodes on which the SFIs are running. The DFM assigns events to the SFIs, load-balancing the farm through a modified round-robin policy where busy SFIs do not receive additional assignments. An SFI can be busy because either the number of outstanding assignments or the number of built events not yet dispatched to the EF reached the respective configurable maximum values, typically 10 for the former and 100 for the latter. The DFM also collects the identifiers of built events and events rejected by L2. In turn, it informs the ROS that the corresponding data can be deleted from the buffers. Using the algorithm described in ref. [69], the DFM also keeps track of the oldest event still to be handled[6]. The ROS can use this information, sent to it as part of the delete information, to activate garbage collection procedures.

[6]Each L2SV sends together with the L2 decision information the L1Id of the oldest event handled by that L2SV for which an L2 decision has not yet been generated. On the basis of this information the DFM can determine the L1Id of the oldest event.

2016 JINST 11 P06008

2.5.1 Event Builder hardware

The DFM and SFI applications are executed in rack-mounted PCs equipped with GbE connections to both control and data networks. The building farm consists of 48 nodes each running 2 SFIs, i.e. 96 SFIs in total. Each node connects via a pair of GbE links to the DataCollection Network and via another pair of GbE links to the BackEnd Network. Each SFI has access to two dedicated connections, one to the DataCollection and one to the BackEnd Network. The nodes are powerful enough to execute two concurrent instances of the SFI without performance penalties.

The ATLAS Event Builder can be supervised by a single DFM. However, 12 dedicated nodes have been installed to allow independent detector slices to be run in parallel,[7] for example when commissioning or calibrating different parts of ATLAS.

The Event Builder farm is able to cope with the design building rate of 3.5 kHz, with an event size of 1.5 MB. Each SFI is in fact able to concurrently saturate both input and output links, operating at an effective throughput of 105 MB/s for a large range of event sizes (3.2). For 1.5 MB events the maximum building rate is therefore about 6.5 kHz.

2.5.2 The SFI

The SFI is a multi-threaded application written in C++. The main tasks of the SFI are collection of the data fragments from the ROS, assembly of full events and subsequent transmission to the Event Filter. The architecture of the application is shown in figure 19. Threads or pools of threads communicate using queues and interact with network cards or, only for testing purposes, with the local disk. Copying of data fragments is minimized, only references to them are exchanged between the threads. The queues allow asynchronous activity of different threads. The event dispatching stage (central box of figure 19) handles complete events. It therefore operates at the building rate per SFI, i.e. at about 1% of the total event building rate for a total of 96 SFIs. The assembly section however, is responsible for the individual ROS fragments, and runs at a rate corresponding to the building rate per SFI scaled up by the number of ROS PCs from which data is received. For physics events (data received from 151 ROS PCs and an L2RH), 96 SFIs and the design event building rate of 3.5 kHz the assembly section rate is about 5.5 kHz.

Following assignment by the DFM, the SFI requests and receives data fragments from the ROS via the DataCollection Network. Owing to the traffic pattern, consisting of many packets potentially sent simultaneously to the same destination, congestion may be caused on the ROS-reply path. The TCP protocol is not good enough to allow full exploitation of the network capabilities in case of congestion. Therefore and as detailed in ref. [71], the SFI implements network traffic shaping (the number of outstanding requests cannot be larger than a certain maximum, which typically is 10),

[7] This can be achieved with the help of the TTC2LAN application. This application runs on the single board computer of each TTC crate containing a "master" Local Trigger Processor (LTP) [9], i.e. an LTP module that is connected to the L1 Central Trigger Processor. For each sub-detector there is such a module. It sends L1 trigger information via the TTC partition(s) (directly and also via "slave" LTP modules if there is more than one TTC partition) associated with the sub-detector (1.4). For test or calibration purposes the L1 trigger information can be generated by the module itself. The TTC2LAN application is an RCD application (2.3.5) making use of a dedicated plugin, the "RCDLTPModule". The application regularly reads the event counter of the LTP module via VME and generates L2 decision messages. These are sent via the network to the DFM node associated with the sub-detector. For each L1Id in the L2 decision message it is indicated whether the event is accepted or rejected. The fraction of accepted events as well as the number of L1Ids per L2 decision message and the frequency with which the event counter of the LTP module is read can be set in the configuration database.

2016 JINST 11 P06008

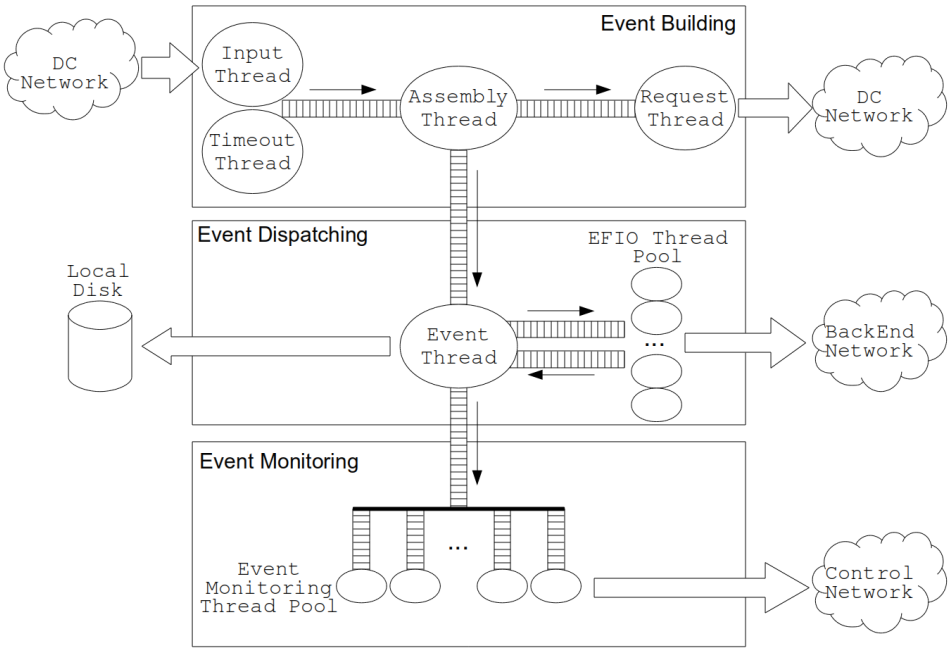


Figure 19. Architecture of the SFI application. Threads exchange pointers to data fragments or events using queues. The thin arrows define the internal data flow, while the boxed arrows show the interactions with external entities. "EFIO" stands for Event Filter I/O, detailed in ref. [70], while "DC network" refers to the DataCollection infrastructure (2.10).

while the destinations of the requests are taken from randomized lists. This ensures successful event building even in case of temporary overload or problems in the network or in the ROS.

Built events are dispatched on request to one of the Event Filter nodes [70] and, after reception is acknowledged, are deleted from the internal buffers. For commissioning and detector calibration purposes, the SFI can alternatively write the built events on the local disk, possibly registering the files in the *SFO-Tier0 handshake* database (2.8.4).

Besides the main event building task, the SFI application also has additional monitoring and data-consistency capabilities. Checks are performed on the built events to verify the synchronization of the front-end electronics and to detect possible data corruption. Furthermore, the SFI can provide a sub-sample of fully built events to monitoring applications, through the emon framework (2.12.1). Several streams of sampled events can be activated, each one serving data with different properties, as defined by the emon selection criteria. As shown in figure 19, the sampling section has been designed with parallel queues and sampling threads to ensure that rare event types are not suppressed in the sampling process.

2.6 Streaming and routing

2.6.1 Event streaming

To allow fast and efficient processing of events recorded by the ATLAS DAQ system and to optimize the turn-around time of offline processing of specific events, the recorded events are organized in so-called "streams" [72]. Three types of events and four different types of streams are distinguished:

2016 JINST 11 P06008

- Calibration events are triggered either by the detector or by dedicated HLT algorithms for specific detector or trigger performance studies. These events typically only require a fraction of the full detector data, corresponding to a typical event size of $O(100)$ kB, and are sent to the calibration stream.

- Physics events are complete events, processed by the TDAQ system without errors, used for physics analyses. They are recorded in the appropriate physics streams. A subset (10%) of these events is also stored in the express stream. This stream is not used for physics analysis, but for checking the data quality, monitoring the status of the detector and of the alignment, and for calibration.

- Debug events experienced a failure during processing by the TDAQ system. They are always recorded in the debug stream and are used for debugging possible problems with the TDAQ system.

The assignment of the event type occurs during the final trigger decision and the information is saved in the event data. An event may be classified simultaneously as a calibration and as a physics event.

2.6.2 Partial event building, event routing and event stripping

Contrary to physics analyses that require the full detector data, the calibration of a given subsystem only requires data from a limited region of the full detector, e.g. for reconstructing a track for alignment of the tracking detectors or for finding an isolated energy cluster for calorimeter calibration. To collect and log such calibration events, so-called Partial Event Building (PEB) and event stripping have been implemented in the TDAQ system. In contrast to full event building for physics events, partial event building does not assemble the data buffered in all the ROBs in a full event. It rather creates partial events from the data retained by a subset of the ROBs. The corresponding list of ROB identifiers is referred to as the PEB list in the following and is filled by an HLT algorithm used for selection of calibration events. The PEB list can either be statically filled, using all identifiers of a given sub-detector, or dynamically, based on the geometrical information of a physics feature in the detector. Once an event is accepted by a calibration trigger it is classified as a calibration event. In the case where the event is a calibration event only, the SFI builds a partial event by taking advantage of the pull protocol in the EB (2.5.2), (4.5.2) and requests only the data fragments specified in the PEB list.

One of the fundamental design principles of the data flow infrastructure is that an event is only built once and no event duplication occurs until the streaming of the data at the SFO. Hence, if a calibration event also passes one of the physics triggers and is thus also classified as a physics event, the complete event is built for storage in the physics streams. Subsequently the so-called event stripping method will be executed to produce the partial event copy required for the calibration stream, i.e. by searching the data fragments in the full event for the ROB identifiers in the PEB list.

Routing of the events through the ATLAS TDAQ system depends on their type, and thus minimizes the resources needed. Event stripping can be executed either by the EFD or by the SFO. The flow of different event types through the TDAQ system is illustrated in figure 20. Events that are only triggered by an L1 calibration trigger are processed by dedicated L2 algorithms. These algorithms do not process any event information, but simply prepare the necessary information for

2016 JINST 11 P06008

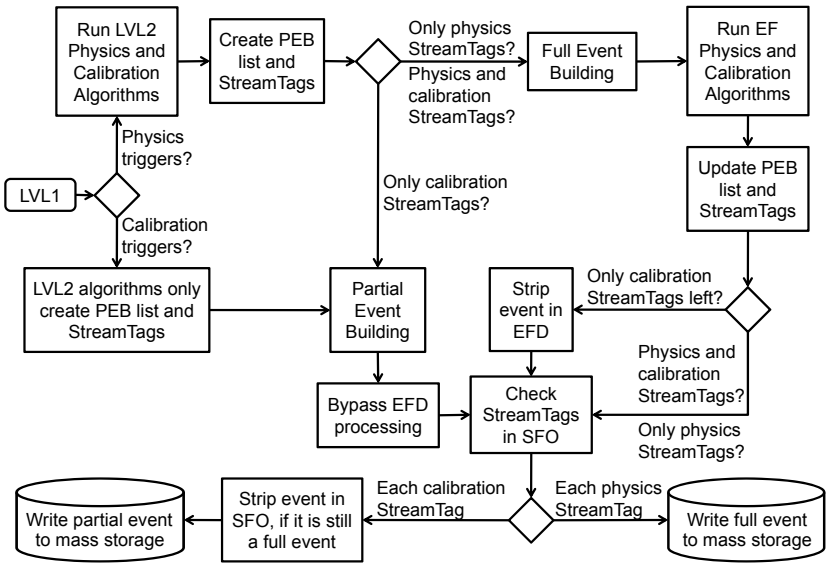


Figure 20. Scheme indicating the flow of events through the TDAQ system.

partial event building and routing of the event. No other algorithms are run on such events and they are routed directly to the SFOs as calibration type events. Events accepted by an L1 physics trigger are processed by the L2 physics and calibration algorithms. If an event is rejected by the L2 physics trigger selection but accepted by a calibration trigger it is partially built and routed to the SFOs as a calibration type event. If it is accepted by an L2 physics trigger it is classified as a physics event, to be fully built by the Event Builder and processed through the EF. For events also accepted by a calibration trigger event stripping may need to be applied before the event can be stored in the calibration stream. For an event rejected by the EF physics triggers, but accepted by a calibration trigger (L2 or EF), the partial event is extracted in the EF using the information from the PEB list, after which it is routed to the calibration stream. An event accepted by an EF physics trigger, as well as by a calibration trigger (L2 or EF), is classified as a calibration and as a physics event. Event stripping can therefore not be performed at the EF since the full event is needed and event duplication is not accepted at the data-flow level. The event is thus transferred to the SFO, where the complete event is routed to the physics streams. In addition the SFO executes the event stripping, using the information from the PEB list, and routes the resulting partial event to the calibration stream. Any event causing an error at a given processing step will be routed directly, without compromising data taking, to mass storage as part of the debug stream.

2.7 The Event Filter

The Event Filter (EF) reduces the data volume so that it can be handled offline, by the mass storage operations and by the subsequent offline data reconstruction and analysis steps. The design target, about 300 MB/s, requires a rejection power of about one order of magnitude.

The EF consists of a farm of processing nodes (at the end of 2011 there were 630 nodes[8]) housed in racks (2.18) and connected to the Event Builder and event storage nodes via the BackEnd switch (2.10.1). The number of processor cores installed at the end of 2011 and the average event size (about 1.5 MB) entails a maximum average latency for the design input rate of 3.5 kHz of about 2 seconds.

To keep the number of network connections managed by every application at a reasonable level each EF node requests events from about 12 SFIs. Each SFI serves an equal number of EF nodes (about 60). These groups of EF nodes have roughly the same amount of processing power, which guarantees load balancing of the EB-EF system and fault tolerance in case of failures of EB or EF nodes. Every EF node can send event data to any of the SFOs, these therefore each manage about six hundred network connections. The EF input/output communication protocol (EFIO) is described in ref. [70]. The SFIs and SFOs act as servers, the EFDs as clients either requesting data from the SFIs or requesting one of the SFOs to accept event data.

To ensure data integrity and fault tolerance, the key paradigm of the EF system design is the decoupling between event processing and data flow operations [75]. In each node, a single data flow process, the multi-threaded EFD ("EF Data flow component"), manages the communication with the SFIs and SFOs and makes the events available to the data processing and event selection applications, the EFPUs. The selection decision and the associated information (EF Result) are communicated back to the EFD. This decoupling provides reliability in case of a crash during data processing and prevents biases in the recorded physics sample[9]. During their transit in the processing node the events are stored in a shared memory, named SharedHeap, which is managed by the EFD. EFPUs have read-only access to the SharedHeap and therefore cannot corrupt the original event data.

The decoupling of data flow and data selection processes also results in scalability and efficient utilization of the processing resources provided by multi-processor and multi-core architectures. Additionally the number of data flow connections is independent of the number of cores, while in each node a single EFD can easily serve hundreds of EFPUs.

2.7.1 The EFD

Local storage: the SharedHeap. The memory used by the EFD for event storage, the SharedHeap, described in the previous section, is reserved at configuration time. A simple and efficient algorithm allows dynamic management of memory blocks of dimensions 2^n bytes[10]. Every incoming event is stored in a new allocated block. The block is freed only when all references to the event are deleted.

Data security is enforced by memory protection policies. The EFPU accesses the events in read-only mode and any write attempt is denied. Not even the EFD Tasks can touch the original event: possible event changes (e.g.: addition of the EF Result fragment) are stored, as differences compared to the main copy, in separate and writable SharedHeap regions.

[8]Equipped with either two Intel X5650 2.67 GHz processors [73], each with 6 cores (one third of the nodes), or with two Intel E5540 2.53 GHz processors [74], each with 4 cores.

[9]Process crashes could be related to a specific event topology and the loss of these events could invalidate the physics result.

[10]The limited spectrum of possible block sizes results in management simplicity and efficient exploitation of the allocated storage space.

2016 JINST 11 P06008

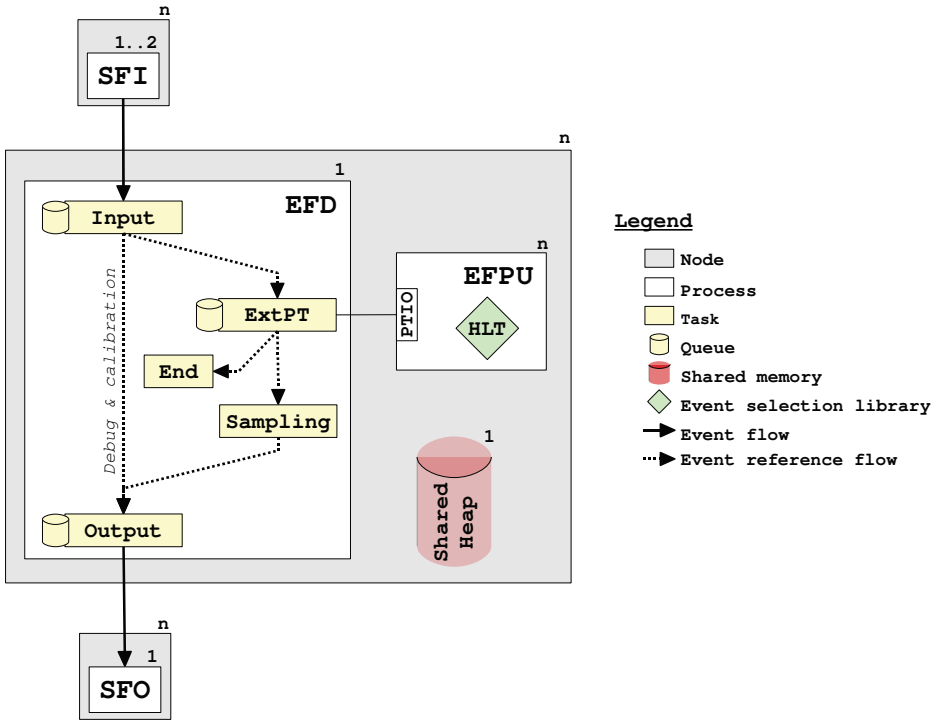


Figure 21. Standard configuration of Tasks within the EFD.

Furthermore, data security is also guaranteed in case of an EFD crash. The SharedHeap is mapped on a local file. Therefore the events can be recovered from the file system if there is an EFD crash, as the operating system itself manages the actual write operations. The system could be out of synchronization only in case of a power cut, OS crash or disk failure, which with high probability will not be related to the event type or topology and therefore does not induce physics biases in the recorded data.

Internal data flow. The EFD is not a monolithic application, but a framework characterized by a modular design, that provides the flexibility required to cope with the technology trends and the evolution of the requirements of the experiment.

Each data flow function is implemented by a specific module, referred to as a Task, which receives events, processes them and forwards them to another Task. Each Task has one or more successor Tasks and therefore they can be dynamically interconnected to form, inside the process, a fully configurable data flow network. A Task may own a dedicated queue to absorb local rate fluctuations. It is executed either by the general worker thread or by its own thread, as is the case for Tasks implementing interfaces to external components (e.g.: SFI, SFO, EFPU), as multi-threading makes it possible to absorb communication latencies.

Once an event is stored in the SharedHeap copying of the data is avoided by passing only references to the event from one Task to another Task via the internal data flow network. These references are implemented as smart pointers providing garbage collection capability. The flow is completely data driven, no data flow manager is needed.

2016 JINST 11 P06008

The Tasks are combined to setup the required data flow paths according to the specification in the configuration database, the standard setup is shown in figure 21.

The Input Task receives fully built events from the SFIs belonging to the sub-farm. One connection is opened for each SFI and each connection is handled by a dedicated thread. The incoming events are stored in the SharedHeap. References to debug type events are directly forwarded to the Output Task, bypassing the EFPU processing stage. The same route is followed by references to events that only belong to the calibration stream. All the other event references are delivered to the ExtPT Task.

The ExtPT Task implements an UNIX domain socket server accessible by all the EFPUs running inside the node. On request of an EFPU the ExtPT Task returns the SharedHeap address of the event to be processed. The HLT decision process produces an EF Result in the form of a ROB fragment that has to be appended to the original event. The fragment is copied to a new allocated region of the SharedHeap, the original event is not touched. Information on the required modifications of the event data (addition of a ROB fragment and update of the header) are stored in the object that represents the event. Possible EFPU problems are reliably handled by the EFD, which can identify their crashes via either socket hang-ups or by means of configurable timeouts. In both cases, the ExtPT Task can re-assign the associated event to another EFPU or forward it to the Output Task.

The stream assignment produced by the EFPU is used to route the events to the successor Tasks. In the standard configuration, two successors are configured. In case of a missing stream type the event is considered as rejected and therefore the reference is forwarded to the End Task, in charge of deleting event references[11]. Otherwise, the event reference is forwarded to the Sampling Task, which forwards all the event references to the Output Task and duplicates some of them for filling a dedicated queue. The latter is visible to the monitoring service (2.12) that can fetch events according to specifiable selection criteria. The queue size and the sampling rate are configuration parameters.

The OutputTask forwards the events to the SFOs. It opens connections with all the available SFOs, each connection is handled by a dedicated thread. The event, which at this stage is scattered across different SharedHeap blocks if it has been processed by the EFPU, is rebuilt on the fly by the Output Task during serialization for output via the network. The Output Task strips the event according to the PEB list available for events classified only as calibration events (2.6.2).

The SharedHeap and each Task have their own set of counters and histograms. A dedicated thread is in charge of gathering these and publishing them in IS (2.2.2) and OHS (2.12.1).

Backpressure management. Under standard running conditions the SharedHeap is almost empty, containing, on average, less events than the number of CPU cores. An increase in the occupancy indicates backpressure from the EFPUs or from the downstream components. In the former case, the processing power is not sufficient to sustain the incoming rate. In the latter case the bottleneck could be the SFOs or the network bandwidth between EFD and SFOs.

To prevent exhausting the local storage resources, the occupancy level of the SharedHeap is used to control the behavior of the Input Task: the event request rate is slowed down proportionally to the SharedHeap usage and is halted if a configurable watermark is exceeded. This mechanism provides smooth upstream propagation of the backpressure and therefore prevents rate oscillations and beating between the EB and EF systems.

[11]The event itself is deleted only if there are no more valid references inside the EFD.

2.7.2 The EFPU

Each EF node hosts multiple copies of the EFPU (also referred to as Processing Task or PT). Usually the number of EFPUs is equal to the number of available CPU cores.

Like the L2PU, the EFPU hosts the trigger selection software, which is executed in a dedicated thread. They share the same software design, the same HLT API and most of the services already described in section 2.4.3. The main difference is the interface with data flow: the EFPU receives the full event and therefore no data collection operations are needed during event selection.

For testing and debugging purposes a dummy version of the event selection software has been developed (ptdummy). It provides configurable CPU usage ("CPU burning") emulating actual HLT processing and allows development and testing of data flow components independent from the HLT project.

The EFPU retrieves event data either from the EFD or from the monitoring service or from the file system, depending on which dynamic library is loaded during configuration. In the first case the library subscribes to the UNIX domain server socket of the ExtPT Task and provides, making use of the required data security policies, access to the data stored in SharedHeap.

2.7.3 EF fault tolerance and error handling

The EF system is designed to be tolerant to faults of any of its applications, if needed events with errors can be recovered off-line. An EFPU crash is managed at the node level and only the local EFD is affected, as discussed in section 2.7.1. The EFD implements the client part of the EFIO protocol and therefore in case an EFD crashes a new EFD instance, created by the run control system, can re-establish at runtime the connections with SFIs and SFOs. The events owned by the crashed EFD can be recovered from disk (from the SharedHeap file) and forwarded to the SFO system.

Unresponsive EFPUs are detected using configurable processing timeouts. Like the L2PU, the EFPU has a dedicated thread that sets a global warning variable when the processing timeout is approaching. At the end of each processing step, the HLT selection software polls the variable and it has a chance to terminate gracefully the processing operation. But if the execution thread is too busy and the variable cannot be polled the timeout expires. In this case the EFD closes the connection, forwards the event to the debug stream and informs the run control system. The latter takes care of killing and restarting the EFPU.

If a network communication error occurs whilst sending an event to a downstream application the SFI or EFD concerned closes the socket and re-assigns the event to another destination (EFD or SFO respectively). The same strategy is applied if communication via the network times out, this deals with network faults or unresponsive EFDs and SFOs. If the timeout occurs while waiting for an event acknowledge message from a downstream application, it is impossible to verify if the latter has successfully received the event. Therefore the upstream application, before re-sending the event to another destination, sets a dedicated duplication warning bit in its header, because the given event might be saved twice.

2.8 Data logging

The data logging farm writes the event data received from the EF to local disks and organizes the events in files based on the trigger streaming decisions (2.6). The data files are then asynchronously transferred to the offline permanent storage facilities.

At the design event rate of 200 Hz, the farm has to handle concurrent input and output throughput of 300 MB/s and to provide enough disk space to locally buffer 24 hours of continuous data taking without offline mass storage support.

During the commissioning and cosmic data-taking periods it became clear that it should be possible to sustain very high throughput to disk, of the order of 1 GB/s, for limited periods and with reduced buffering capabilities. These special running conditions are needed to accumulate large amounts of data for trigger commissioning or detector studies and required an upgrade of the hardware, which took place in the second quarter of 2010.

2.8.1 The data logging farm

The data logging farm (figure 22) is composed of 6 servers each equipped with 3 PCIe RAID interface cards and 24 hard disks of 500 GB. The design choices behind this architecture are discussed in ref. [76]. Five of the nodes are used for data taking. Compared to the system presented in ref. [76] the current farm nodes have been upgraded, as mentioned. The upgraded nodes contain in particular a SuperMicro X8DTE-F-O motherboard [77], two Intel Xeon E5520 2.26 GHz quad-core CPUs [78], three Adaptec 5805 RAID cards capable of supporting RAID5[12] [79], and 6 GbE ports, respectively 2 on the motherboard and 4 on an Intel 1000PT quad-port PCIe card [80].

At the application level, the hardware resources are represented by three 3.5 TB disk volumes corresponding to the three RAID cards, a GbE link connecting to the Control Network and two 2 Gbit/s Ethernet links (each realized by means of bonding two GbE links [81]), one connecting to the BackEnd switch and the other to the Castor switch.

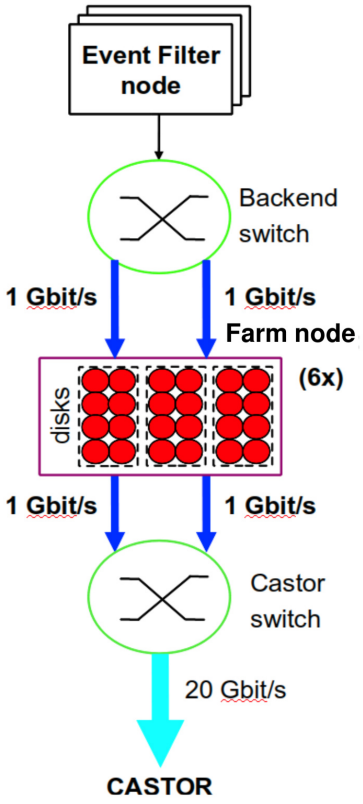


Figure 22. Hardware architecture and connectivity of the data logging farm.

2.8.2 The SFO

On each node of the data logging farm one instance of the SFO, a multi-threaded application written in C++, is running. The SFO receives event data from the EF nodes and stores them into files on the three local disk volumes. Given the limited number of farm nodes and the need for redundancy, a sub-farm structure is not acceptable at this level. Each EF node can therefore send event data to any of the SFOs. This implies that the SFO communication library must be able to efficiently handle $O(10^3)$ concurrent connections, which is achieved using a configurable pool of communication threads that serve the incoming requests in parallel.

To obtain the best I/O performance from the RAID volumes concurrent write and read operations need to be avoided. Each SFO therefore writes to a single volume at a time and cycles over

[12] RAID5 is a striping RAID mode with parity data distributed on all the disks.

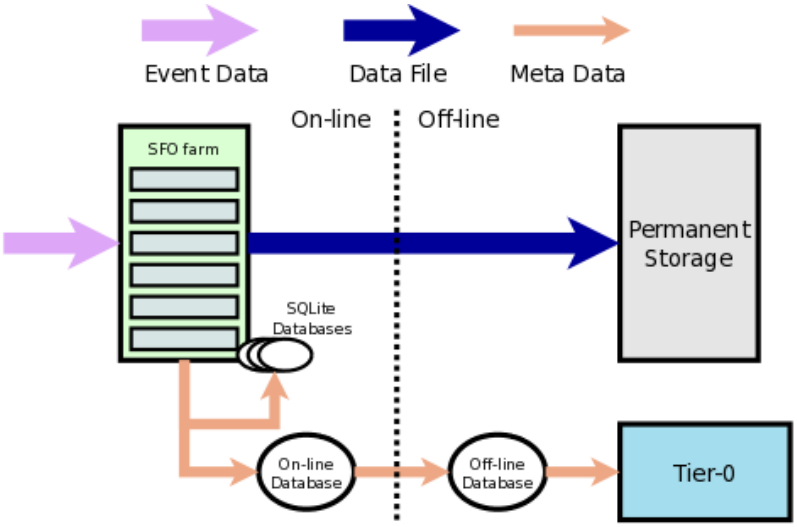


Figure 23. Event data and meta-data flows between the ATLAS online and offline facilities.

the available volumes in a pure round-robin mode. The volumes not used by the SFO are available for data readout by the Castor script, which is described in the next section.

The SFO organizes the data in a set of files on disk, based on the event streaming properties, each file corresponds to a unique stream and luminosity block. If necessary it applies event stripping (2.6.2, figure 20).

2.8.3 The Castor script

The Castor script is the second software component of the data logging system. The script, implemented in Python, is complementary to the SFO. It transfers the data files to the CASTOR mass storage system [82] at the computer centre of CERN and subsequently deletes them from the local volumes.

The script is flexible and extensible in terms of file transfer configuration and delete policies. It is therefore used for transferring various files, not just those from the data logging, from the ATLAS online infrastructure to CASTOR, with up to 40 instances executing at any time. The scripts thus transfer files with many different types of contents, ranging from archived monitoring information to stand-alone sub-detector calibration data.

2.8.4 SFO-Tier0 handshake

The data logging farm is expected to produce and transfer of the order of 5 million files per year. A dedicated mechanism, based on shared database tables, has been developed to guarantee correct bookkeeping and persistence of file meta data, to allow data integrity checks and to seed offline processing of the raw data files.

The file meta data stored in the so-called SFO-Tier0 handshake tables includes file sizes, checksums, file locations, writing and transfer statuses as well as run numbers and streaming information. Both the SFO and Castor script access the database and verify that the life cycle of each file, from creation to deletion, is correctly represented in the tables.

2016 JINST 11 P06008

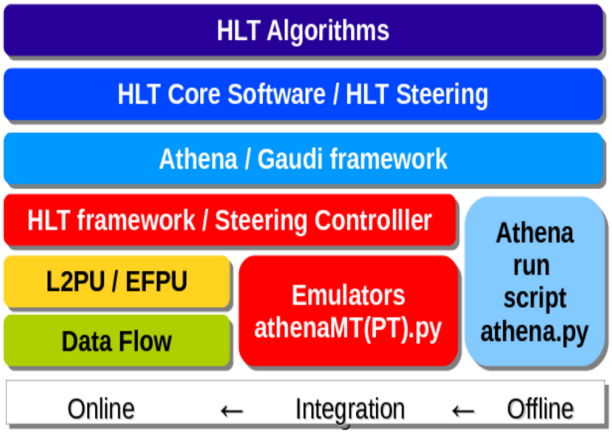


Figure 24. Software layers and domains of the ATLAS High-Level Trigger. Each software layer has abstract interfaces that ensure portability between the offline, integration and online domain.

The flow of both data and meta data between the online and offline facilities is shown in figure 23. Data are pushed from the data logging farm via two paths to the offline containers: the raw data files to the CERN mass storage infrastructure and the meta data to the online database cluster. The online database tables are automatically copied to the offline cluster using Oracle streams (2.2.4). In case of offline service failures, raw data files can be buffered on the SFO disk volumes, while the online database cluster can retain the meta data. However, given the importance, and uniqueness in some cases, of the file meta data, a copy is also maintained in local SQLite [44] files on each node of the data logging farm. This allows the online database tables to be re-populated a posteriori if there is a failure of the online cluster.

2.9 HLT integration of online and offline software components

As shown schematically in figure 24, the HLT software is used in three domains: online, integration and offline. The integration domain is used to begin the validation of new software for online use. Each domain has its own specific requirements, use cases and applications that run HLT algorithms. To cover this wide range of use cases the HLT software is organized in layers. The HLT Steering [83, 84], which is part of the HLT Core Software, controls the HLT algorithms. Both the HLT algorithms and the HLT core software adhere to the architecture of the offline software framework Athena [85], which itself is based on the component framework Gaudi [31]. Abstract interfaces ensure that each component can be replaced without changes in the client code. In the HLT certain standard services are replaced by online specific implementations. The result of the algorithm execution is communicated in the online and integration domains to the underlying HLT framework, while it is forwarded to the reconstruction and analysis software in the offline domain. The athena.py script is used to run the offline reconstruction and analysis programs with the HLT code. In the integration domain the L2 and EF emulator scripts athenaMT.py and athenaPT.py (2.9.3) and in the online domain the L2PUs (2.4.3) and the EFPUs (2.7.2) execute the same reconstruction and analysis code using the HLT framework.

2016 JINST 11 P06008

2.9.1 HLT software

The trigger selection proceeds in steps which are defined in the trigger menu. A menu consists of up to several hundred trigger chains, where each chain defines the L1 seed and HLT selection for a single physics signature, such as an electron or a muon with a large transverse momentum. A chain consists of multiple steps, with one or more HLT reconstruction and selection algorithms executed for each step. The HLT Steering part of the HLT Core Software schedules the HLT algorithms corresponding to the input seed and the menu prescription, so that the correct algorithms and selection criteria are used. Event specific quantities are passed between HLT algorithms as C++ objects, which are defined in the Event Data Model (EDM [83, 86]). These objects are posted by HLT algorithms to data managers [87], which allow HLT algorithms later in the chain to retrieve and further analyze these data. In case multiple chains need to run the same algorithm using the same input data, the results from the first execution are automatically cached to minimize the data processing time. Further platform and storage technology specific details of the data access are hidden from the algorithms by using different converter functions for different data retrieval and storage technologies. In the online domain the HLT framework interfaces the HLT Core Software (common to L2 and EF) to the online run control and data flow software. The Steering Controller [88], part of the HLT framework, completely controls the HLT framework in terms of finite state machine transitions and management of the event loop (see figure 25 for the case of L2). It also provides access to the online configuration system and to event data, either directly from ROBs for L2 or from full events for EF. Furthermore it handles error conditions arising from algorithm execution in the online data flow context and reports them back to the L2PU or the EFPU. After the event selection code has finished processing the event, it packs the detailed event decision record into a raw data fragment and forwards it, together with the event streaming information, to the online data flow software.

The common software environment for the HLT software, provided by the HLT framework and by the Athena run script, allows for efficient reuse of code from a large software base developed by many contributors, and an offline like environment for trigger algorithm development. The common code base for the online and the offline software guarantees the reliability of trigger performance evaluations, which are mostly performed in an offline setup. Examples of re-utilized components are the storage manager, the Event Data Model, the detector description, the services for handling conditions data and many reconstruction tools, which were developed by the offline community. Only the HLT Steering and certain specialized trigger algorithms are HLT specific. The HLT framework is also based to a large extent on the Athena architecture and re-uses whenever possible core offline services online.

A key feature for providing transparent access mechanisms to data for offline and for online processing is the name based service architecture of Gaudi. It allows implementation of services with different online back-ends and configuration with the same name as used in the offline domain. An example is the access to data fragments in the ROBs for L2, where a special online ROB data provider service directly contacts the ROS, retrieves the required fragments and presents them to the selection algorithms as if they had been read from a complete event in the offline domain. Other services read selected detector conditions data from IS and present them to the algorithms as if they would have been retrieved from the offline conditions database. This is shown schematically in figure 26. In a similar way a special implementation of the Gaudi histogram service allows the algorithms in

2016 JINST 11 P06008

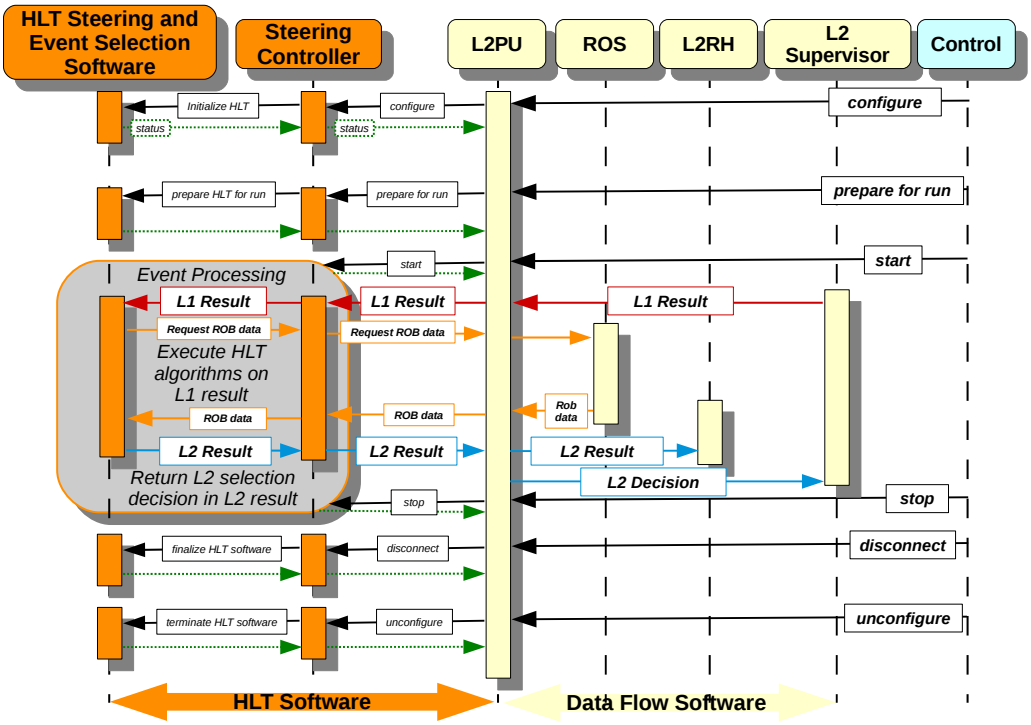


Figure 25. Interactions of the HLT steering controller and the HLT event selection software with the data flow applications and the run control in L2. Only state transitions relevant for HLT are shown.

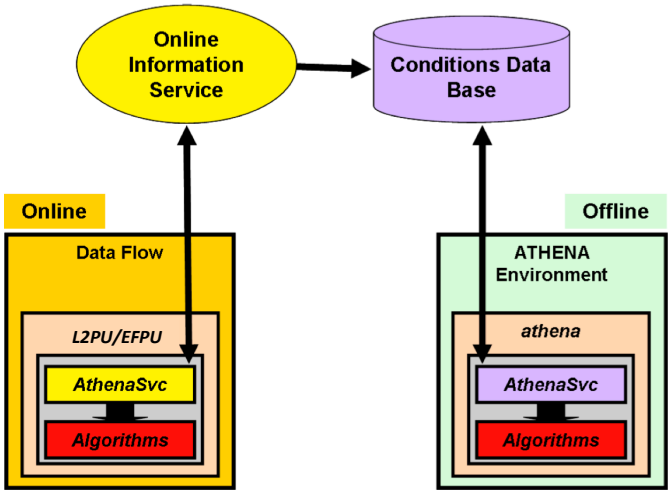


Figure 26. Schematic view of an Athena service that reads online specific detector conditions from the online Information Service (IS) and presents them to the event selection algorithms in the same format as the corresponding service does in the offline domain. There the data are read directly from the offline conditions database. In this way algorithms have complete transparent access to their conditions data. The arrow between IS and the offline conditions database indicates that selected data delivered by IS are also archived asynchronously in the conditions database.

2016 JINST 11 P06008

the online domain to dispatch their histograms to the TDAQ monitoring and histogram collection infrastructure, whereas offline the standard service will save the histograms to a file. A special case is the online job configuration service [89, 90], where the complete trigger configuration is read from a database instead of from Python job setup scripts, as would be the case for offline processing.

2.9.2 Real-time configuration changes and timeouts

The output rate of HLT selection chains can be controlled with so-called prescale factors. These can be changed during run time at luminosity block boundaries [91, 92]. Similarly, the update of beam spot parameters for HLT track reconstruction algorithms can be requested at luminosity block boundaries [93–95]. In both cases the update is signaled by a field in the L1 Central Trigger Processor (CTP) data fragment that indicates the luminosity block number at which a pending change should become active. For the prescale changes, the update is triggered manually, while the updates of the beam spot parameters proceed fully automatically as explained below. CTP fragments are distributed either via the RoI Builder and the L2SVs for L2 as part of the RoI requests, or via the CTP ROD and the Event Builder for the EF as normal event fragments. The Steering Controller decodes the field and initiates the lookup to the trigger database (or conditions database in case of the beam spot) that holds the updated configuration keys (or parameters) for the new luminosity block. The parameters for the updated keys are read by the HLT Steering and applied to the HLT selection. A system of database proxy applications [45] ensures that the approximately simultaneous database access of the HLT processes does not give rise to long delays.

The updates of the beam spot parameters are not only relevant for the HLT tracking algorithms but in fact critical for trigger algorithms that impose requirements on track impact parameters or decay lengths as these rely on the precise knowledge of the interaction point. The observed variations during a fill, or those from fill to fill, are often larger than the sensitivity of these algorithms. To track such changes, a dedicated algorithm is executed as part of the L2 trigger that reconstructs primary vertices and publishes histograms of their distributions. These histograms are then aggregated across the farm (2.12.1) and analyzed by a separate application for every luminosity block. A resolution correction is applied that is derived online based on the observed displacement of split vertices. Through fitting, the position, size and tilt angles are extracted. The parameters that correspond to each luminosity block are loaded into the conditions database for further monitoring. In case of the HLT parameters, a sliding average over the last 10 luminosity blocks is used to increase the statistical precision without compromizing the time resolution. A separate module then compares this set of parameters with the ones currently stored for the HLT. Changes are reported when (a) the x, y or z position of the centroid of the luminous region has moved by more than 10% of the corresponding luminous width; or (b) the luminous width has changed by more than 10% in either dimension; or (c) the error on any of the parameters has decreased by more than 50%. Whenever such a change is detected, the parameters are published to IS (2.2.2), and a command is sent to the CTP to instruct it that an update should be performed with the next luminosity block. Upon this command, the CTP fetches the parameters from IS, stores them in the conditions database, and then updates its data fragment with the relevant luminosity block number as described above. The same mechanism that is used for the prescales completes the feedback loop on the HLT processors as they receive the data fragment.

2016 JINST 11 P06008

A slight complication to the updates of the parameters describing the luminous region are the potentially large variations that can take place between fills. These require a bootstrap of the parameters at the beginning of every data-taking run. To initiate this, another controller flags - whenever stable beams are lost - the current set of parameters with a particular status word that allows the most sensitive algorithms (i.e. *b*-tagging) to inhibit their execution for the first few luminosity blocks until the new interaction point has been successfully determined. The entire feedback loop is fully automatic and results in a few updates near the beginning of a run, and a few more over the course of a fill that adjust for orbit or RF-phase variations as well as the typical emittance growth of the two crossing beams.

A limit is imposed on the processing time, since beam conditions or noisy detector modules may cause an event to have a very large number of detector hits leading to excessive online reconstruction times. Timeouts are detected by means of a special thread, they are handled as described in section 2.7.3, see also ref. [96].

2.9.3 Software development model

Code developed in the offline environment can directly be downloaded in binary form to the HLT processors since the interfaces in the offline domain are identical to those in the EFPU and the L2PU environment. Testing of HLT code is less straightforward since it runs inside the L2PU and EFPU data flow applications and requires a data flow environment suitable for testing ("online domain") or that the L2PU or EFPU applications are emulated ("integration domain"). Setting up a data flow environment is possible on a single processing node as well as on a multi-node system. For emulating the L2PU or the EFPU application two command line applications are available, respectively athenaMT.py [97] and athenaPT.py [98]. They are written in Python and share to a large extent a common code base. Differences arise only in the way the HLT event selection software accesses raw data and in L2 and EF algorithms. To simulate the raw data access via network requests for L2, athenaMT.py loads the raw data fragments for each event into memory and provides them on algorithm request to the event selection software. In a similar way athenaPT.py provides the full raw event to the HLT code. Both emulators allow interactive cycling through the ATLAS trigger finite state machine to test if newly created code is compatible with trigger operation. Command line options and debug aids for the emulators are similar to those of the offline Athena run script. Therefore developers with an "offline background" can more easily get acquainted with HLT development, while they do not need to be familiar with detailed technical aspects of the data flow software and furthermore they are shielded from changes in it. A large variety of tests can be performed with the command line applications, but of course the final certification of the HLT software has to be done on a large distributed system.

2.9.4 The AtlasTrigger and AtlasHLT projects

The software packages which belong to the HLT framework form the so-called AtlasHLT software project whereas the AtlasTrigger project holds the HLT Core Software and the HLT algorithms. These software projects are structured like other offline software projects [99] and use the same version control, build and testing tools. Because of its nature as an interface between data flow and offline, the AtlasHLT software project is the only ATLAS software project which directly depends on data flow software and on all other ATLAS software projects. Therefore the releases of the AtlasHLT project have to be coordinated with trigger and data acquisition (TDAQ) releases and

2016 JINST 11 P06008

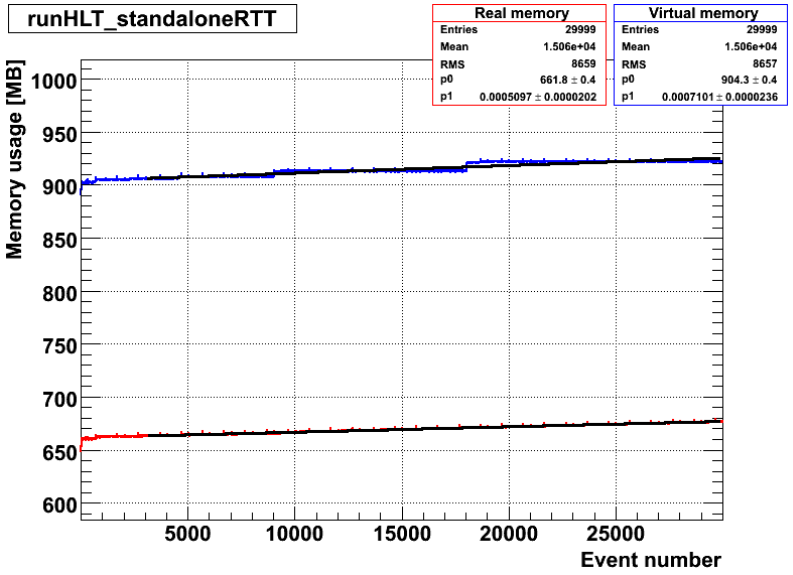


Figure 27. Example of an automatic memory leak test for L2 code run every night after the AtlasHLT developer builds. The graph shows the virtual (upper curve) and real (lower curve) memory usage for the tested framework and selection code components as a function of the number of processed events.

with offline releases. Further dependencies exist due to the common use of *LCG* software [100] and the common software project tdaq-common, which mainly hosts the code for the ATLAS raw event data format [20].

Developers register their software components with the ATLAS Tag Collector and every night a new test release is built. Also every night automatic code tests are launched using mainly the emulator applications athenaMT.py and athenaPT.py. The test suite is complemented by automatic code tests run on a single node data flow setup. Figure 27 shows an example of the results of a memory leak test, performed with the release test suite. Since the offline code modules are executed in the trigger orders of magnitude more often than in the offline reconstruction farms, testing of code robustness and careful control of memory leaks is of great importance. For example performance targets for the trigger require memory leaks in L2 to be smaller than 10 bytes/event and to be less than 1 kB/event in the EF. Therefore performance monitoring tools have been developed which help to follow the evolution of execution time and memory usage of the code over the different release cycles in the offline, the AtlasTrigger and the AtlasHLT projects.

Validating and deploying a new release is time consuming and storing many, almost identical, releases requires much disk space. To allow for fast bug fixes and the addition of minor new features, a so-called cache named AtlasP1HLT has been introduced. This can contain newer versions of any of the software packages in AtlasHLT, AtlasTrigger or other offline software projects. These will override the existing versions in the dynamic link order. In this way, the underlying code can be changed, but since only a few packages are recompiled, no changes can be made to public interfaces. The cache uses the same software management and build infrastructure as full releases, but the contents are tightly controlled to ensure that the performance is not accidentally degraded. If required, a simple bug fix can be deployed using this mechanism in less than 24

2016 JINST 11 P06008

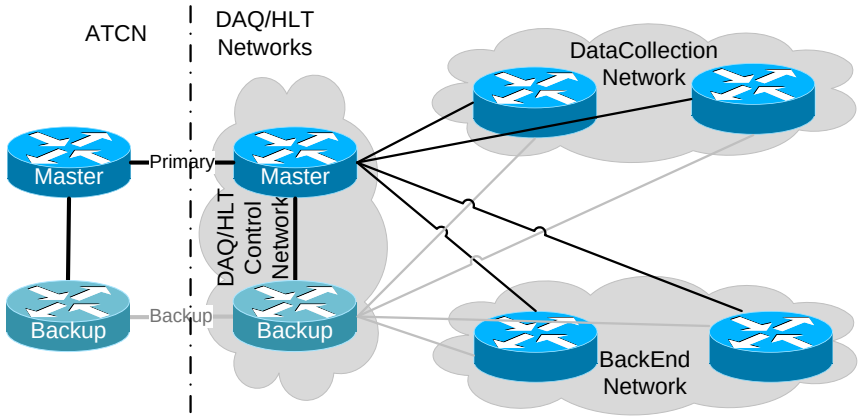


Figure 28. The DAQ/HLT routers (redundant routers and links are shaded).

hours; for new features and other non-trivial changes additional time is spent on testing. The same automatic tests as used for the AtlasHLT project are used for the AtlasP1HLT cache and catch most problems. In addition, before installing a new version of the cache in the online system, $O(10^6)$ previously recorded events are processed through athenaMT.py, athenaPT.py and the offline event reconstruction. Besides testing the robustness, the results are also compared against those of previous processings and all differences are required to be understood before a cache is signed off for online deployment. After installation in the online farm the new cache is deployed if running a small test partition, as described in 2.16, is successful.

2.10 Networking

The DAQ/HLT system comprises a control network, which provides infrastructure and operational services, as well as two dedicated data networks, used exclusively for transferring the event data (2.1.3). All these networks rely on the IP (Internet Protocol) protocol and are implemented using Ethernet technology.

This section describes the complete networking architecture solution for the ATLAS DAQ/HLT system, as well as the framework used for operating it.

2.10.1 Architecture

The number of end-nodes to be interconnected is large and most of them have modest bandwidth requirements [30] compared to their GbE interfaces, therefore typically the nodes in a single rack are connected to "pizza box" switches acting as concentrators. These switches are connected via GbE or 10GbE (10 Gbit/s Ethernet) up-links to the network core, composed of chassis-based devices. End nodes hosting applications with high bandwidth requirements have direct GbE connections to the network core.

During data-taking periods the TDAQ system must operate round the clock. To address this high availability requirement, the network core has been designed to be resilient. The core devices have a high degree of built-in redundancy, and multiple devices are used to provide alternate paths through the network core [101].

2016 JINST 11 P06008

Figure 28 shows the three DAQ/HLT networks and their interface with the ATCN (ATLAS Technical and Control Network, the general purpose network at the ATLAS experimental site). The DAQ/HLT Control Network has two core routers, working in a master-backup configuration. The master and backup routers are connected to the master and backup routers of the ATCN core respectively. All the connections between these four routers are high speed links (either 10GbE or LAGs[13] of 10GbE links). During normal operation, the backup routers don't handle any traffic, but they can take over the entire functionality in case of failure of a master router.

The cores of the two data networks (DataCollection Network and BackEnd Network) connect to both DAQ/HLT Control routers. GbE links are used for these connections since they serve management and monitoring purposes only.

Both the ATCN and the DAQ/HLT networks use the Open Shortest Path First (OSPF) protocol [103] for internal IP routing. For simplicity, static routes are used for interconnecting the two domains (ATCN and the DAQ/HLT Networks).

DAQ/HLT Control Network. Run control for DAQ/HLT applications is provided through the Control Network (see figure 5). For practical and geographical reasons, the Control Network is split into two parts: the DAQ/HLT Control Network, providing connectivity for DAQ/HLT nodes installed in the barrack in the SDX1 building (2.18.2), and the ATCN, providing connectivity for all other subsystems, e.g. ROS PCs and detector equipment located underground. This section describes the implementation of the DAQ/HLT Control Network.

The network core of the DAQ/HLT Control Network (see figure 29) is implemented with two redundant chassis devices, interconnected by a high speed link (two aggregated 10GbE links). A few tens of infrastructure servers providing control and monitoring services (2.11), (2.12) are connected directly and redundantly to the core with copper GbE links. The rest of the nodes (e.g. nodes in XPU and EF racks) are clustered at the rack level using concentrator switches, with a copper GbE up-link connecting each concentrator switch to each core device.

Since there are no demanding performance requirements, and the network comprises a large number of end-nodes, it is operated at OSI (Open Systems Interconnection model) layer 3, with one sub-net per concentrator switch or group of infrastructure servers.

The Virtual Router Redundancy Protocol (VRRP) [104] is used to provide redundancy without load balancing: for each of the sub-nets a virtual router is emulated by the two physical core routers. In case of the failure of the up-link to Core 1 (see figure 29) of any given rack (say A) the nodes in the rack can still communicate with the nodes in any other rack (say B) by using the following paths: A to B using *rack A switch — Core 2 — rack B switch* and B to A using *rack B switch — Core 1 — Core 2 — rack A switch*. Thus the link between the two core routers (on which OSPF is active) will be used in case of failure of a rack switch up-link.

The servers directly connected to the routers have two physical interfaces bonded [81], using the active backup mode (only one link is active, only if it fails does the other link take over).

DataCollection Network. For improved resiliency and scaling, the DataCollection Network (depicted in figure 30) is implemented using a core made out of two chassis switches interconnected through a high speed link (four aggregated 10GbE lines).

[13] Link Aggregation Groups [102].

2016 JINST 11 P06008

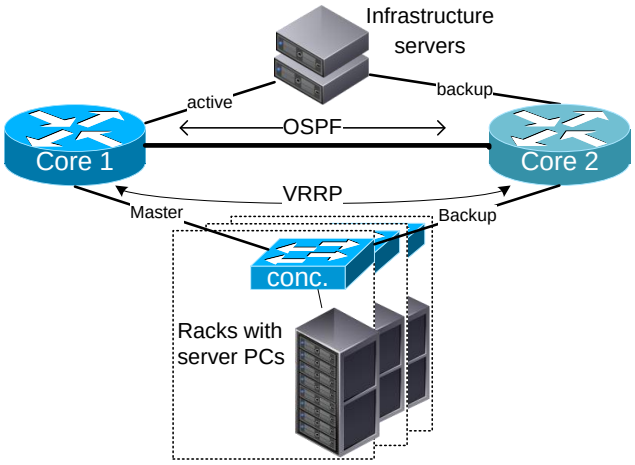


Figure 29. The DAQ/HLT Control Network.

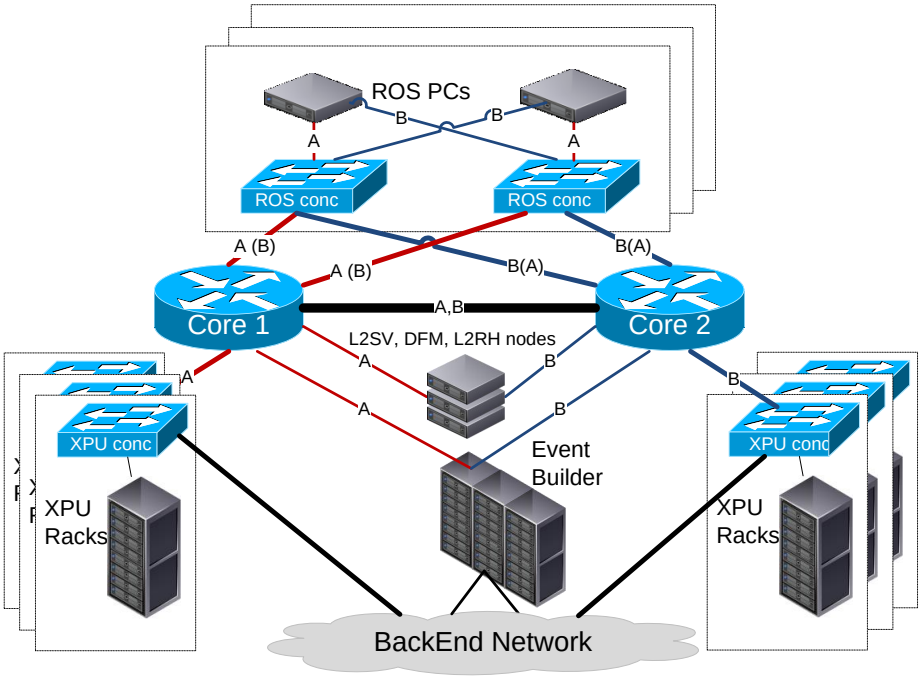


Figure 30. The DataCollection Network.

The XPU nodes (2.1.3) are concentrated at the rack level, while the Event Builder and the nodes running the L2SV, DFM or L2RH applications connect directly to the core. Since the distance between the ROS PCs and the core exceeds the 100 metre range of copper GbE, a layer of "concentrator" switches has been introduced between the ROS PCs and the core: each ROS is equipped with two GbE interfaces that connect to different concentrator switches for improved redundancy. To avoid congestion, each of the two 10GbE up-links of a concentrator switch aggregates traffic for at most ten ROS interfaces.

Because of the demanding performance requirements the DataCollection Network is operated at OSI layer 2. Virtual LANs (VLANs) and the Multiple Spanning Tree protocol (MST) [105]

2016 JINST 11 P06008

are used to provide redundant connectivity with load balancing over the up-links of the ROS concentrator switches.

Each core switch implements a separate VLAN (VLAN A for Core 1, and B for Core 2, see figure 30), and each VLAN is mapped onto a separate MST instance (MSTi). In figure 30 each link carries a label indicating the VLAN it is part of and its state in the respective MSTi: label "*A*" means the link is part of VLAN A, and its state is forwarding in MSTi A; label "*A,B*" means the link is part of both VLANs, and its state is forwarding in both MSTi instances; label "*A(B)*" means the link is part of both VLANs, and its state is forwarding in MSTi A and blocking in MSTi B and vice versa for "*B(A)*".

In case of an "*A(B)*" up-link failure, the "*B(A)*" up-link of the same switch will transition to forwarding in MSTi A too (its label will become "*A,B*"), and the switch will remain reachable in VLAN A through the following path: Core 1 — "*A,B*" inter-chassis link — Core 2 — "*A,B*" up-link.

XPU racks. Two VLANs are defined on all XPU rack data switches: a VLAN for the DataCollection Network (VLAN A for switches attached to Core 1 and VLAN B for those attached to Core 2, see figure 30), and the default VLAN for the BackEnd Network (2.10.1). While the XPU nodes in the racks connect to ports which are part of both VLANs, the up-links are part of a single VLAN (VLAN A or B for the DataCollection up-link, and the default VLAN for the BackEnd up-link). Thus a single switch is used to emulate two separate "virtual" switches, one connecting the XPU nodes to the DataCollection Network, and the other to the BackEnd Network. This allows for fast migration of HLT processing power from the L2 to the EF system or vice versa, via changes in the configuration database that become active at the "configure step" prior to starting a new run (2.11.5).

BackEnd Network. Initially the core of the BackEnd network consisted of a single chassis device. Though the device had a high degree of redundancy (power supplies, switching fabric, management modules) it was a single point of failure, therefore a second device has been added at the beginning of 2012, as illustrated in figure 31.

The Event Filter processing nodes are concentrated at the rack level. The rack level concentrator switches are connected to one of the two cores through a 2 Gbit/s LAG or a 10GbE up-link, depending on the generation and power of the computers installed in the rack. The Event Filter racks are uniformly distributed over the two core devices. The Event Builder nodes and the nodes of the data logging farm require a higher throughput and are connected directly to both core devices.

Similar to the DAQ/HLT Control Network, the BackEnd Network is operated at OSI layer 3 (IP routing at the core, with one sub-net per concentrator switch) to restrict Ethernet broadcast domains to the rack level. Link aggregation [102] is employed not only for the up-links of the older generation processor racks, but also for the connectivity with the Event Builder and data logging nodes, which easily over-load single GbE links.

2.10.2 Network management

The management of the DAQ/HLT networks is done using a mixture of commercial software and tools developed in-house. With respect to the ISO FCAPS[14] management model, only *configuration, fault* and *performance* management are implemented. Because of the nature of the DAQ/HLT networks, there was no need to address the *accounting* and *security* management to date. This

[14]Fault, Configuration, Accounting, Performance and Security Management.

2016 JINST 11 P06008

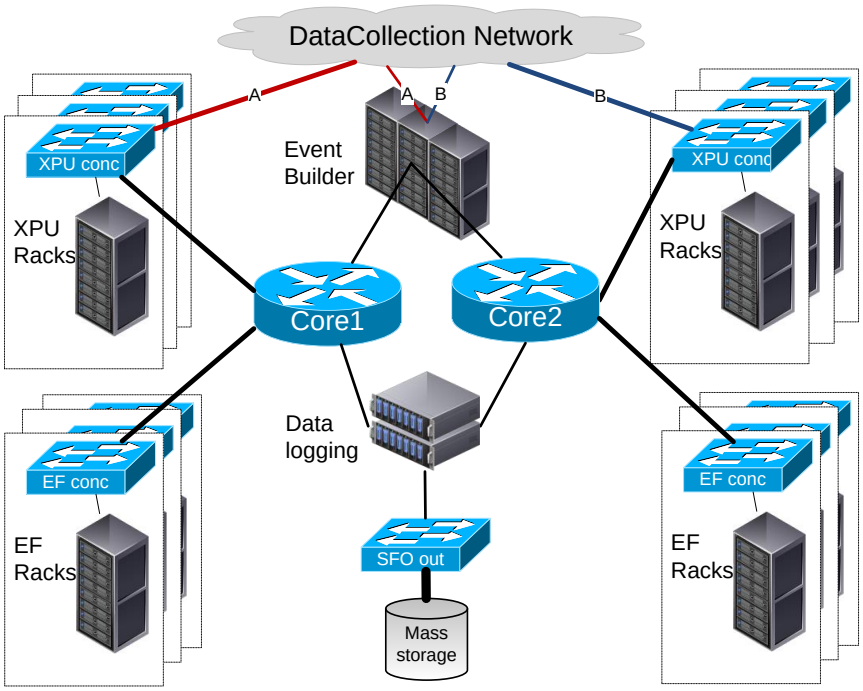


Figure 31. The BackEnd Network.

section will provide an overview of how the networks are operated and monitored, these topics are covered in more detail in refs. [106] and [107] respectively.

Configuration management. A Python module has been developed to provide unified access to the command line interfaces of the switches, regardless of the specifics of each switch. Scripts have been developed making use of this module. These can perform configuration changes and can take care of regular backups of the configuration files of all the network devices as well as of bulk firmware upgrades of the switches.

A coherent logging system is important for troubleshooting and understanding the chronology of the generated messages. Thus, the devices have been configured to have a synchronized clock (using the Network Time Protocol — NTP) and to log messages to a common syslog server.

Fault and performance management. Figure 32 depicts the collections of tools employed for monitoring the DAQ/HLT networks health and performance.

CA Spectrum [108] has been chosen for Fault management. A strong argument for the choice was the compatibility with the CERN wide network operated by CERN's IT department. A Network Service Gateway (NSG) has been designed to enable information from Spectrum to be easily exported to external clients. It caches the topology of Spectrum's network model, and provides an efficient interface, making use of Thrift [109], to external clients. The online expert system (2.11.4) subscribes to the NSG to receive alarms generated by Spectrum. A database (known as the central database) is used to store information from various sources and supply it on demand to several presentation clients. This modular system [106, 110, 111] has been developed to overcome the scaling issues of the aforementioned commercial package when performing high rate network

2016 JINST 11 P06008

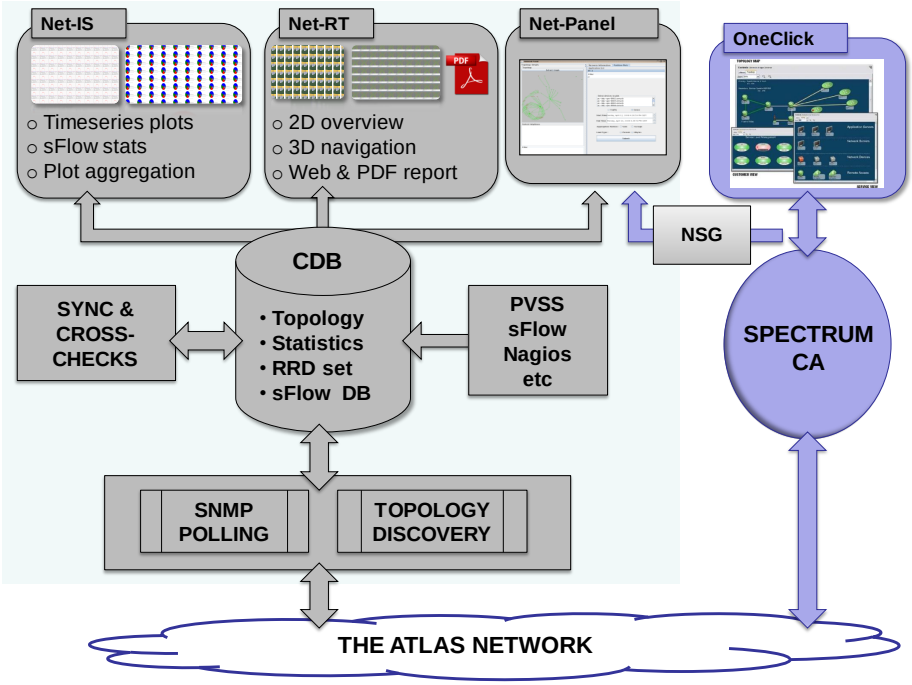


Figure 32. Block diagram of the ATLAS DAQ/HLT network management framework.

statistics gathering (imposed by the real-time nature of the DAQ/HLT system). Time-stamped data are typically stored in round-robin databases (RRD [112]) and consist of network traffic statistics, as well as various environmental and PC specific statistics (2.14.2).

The network specific data sources are an SNMP (Simple Network Management Protocol) polling engine (apoll), a topology discovery engine and a flow analysis engine. The SNMP polling engine efficiently gathers statistics from all the network ports and stores the results in RRD files, with a polling interval of at most 30 seconds. The flow analysis engine, based on sFlow [113] stores statistically sampled flow data. This proved to be extremely useful for a posteriori troubleshooting.

Several flavors of presentation clients have been designed. Some of them are targeted mainly at the operator in charge of monitoring and control of the networking functionality, while others provide in depth information typically accessed by experts. The most complete of them is Net-IS (the Integrated System for Performance Monitoring of the ATLAS DAQ/HLT Network [110, 111]): a powerful interface which provides convenient access to all the data stored or referenced by the central database. The information can either be browsed by several predefined hierarchical trees, or it can be directly searched using regular expressions. Multiple time series can be displayed both overlaid on the same plot (see figure 74 for an example) and as an array of "mini-plots".

2.11 Configuration and control

2.11.1 Overview and architecture

The Configuration and Controls services provide support for a flexible description and smooth control of the distributed TDAQ system. They are designed following a layered component model as presented in figure 33: at the very bottom are external packages and common base libraries

2016 JINST 11 P06008

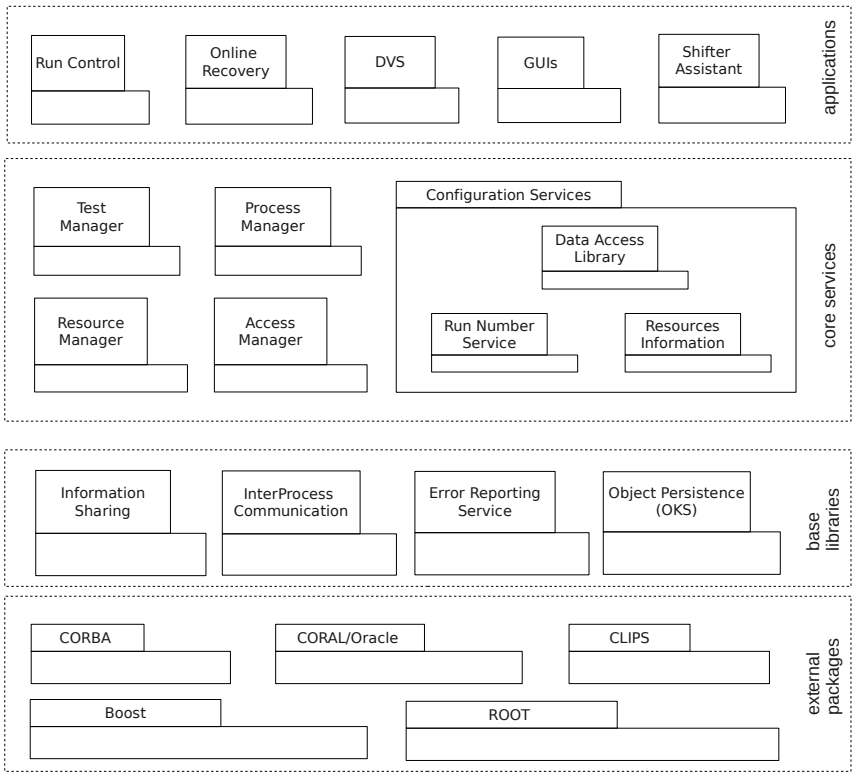


Figure 33. Component architecture of the Configuration and Controls services.

(described in 2.2), higher up is a service layer, including the Configuration Database services and the Process Manager. The application layer contains the Run Control, the Diagnostics and Verification System, Online Recovery system, the Shifter Assistant and also a number of Graphical User Interface (GUI) applications.

The Configuration and Controls components are used by many other TDAQ subsystems and they are designed in a framework approach, allowing the extension of the core functionality by the subsystems.

The central part of the Controls system is the Run Control framework, allowing synchronous execution of data-taking actions by all elements of the distributed TDAQ system.

Scalability and reliability aspects were taken into account when the design of the components was developed. To maximize the run efficiency of the experiment and to store the expertise of developers gained through the years of the experiment, the TDAQ Control system includes advanced verification, diagnostics and complex dynamic error recovery tools, based on an expert system.

In the following sections a detailed description of the design choices and important aspects of implementation of the Configuration and Controls components are presented.

2.11.2 Core services: access, resource, process management

Access management. The Access Management (AM) system [114] is based on the Role Based Access Control (RBAC) paradigm: the access decision for a user is based on the roles the user has in the operation (e.g. shifter, expert) at a given moment. The access rights are grouped by role name

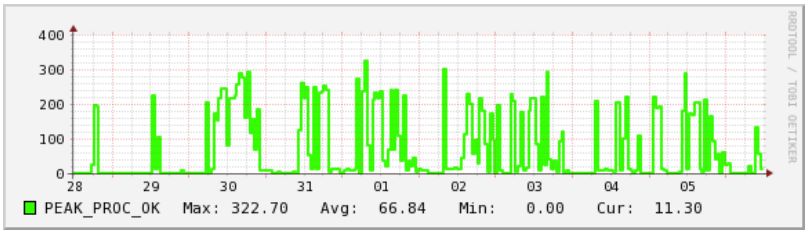


Figure 34. Requests per second as function of time for one of the AM servers during a period of a week (the numbers along the x-axis indicate the day of the month).

and the access to a resource is granted only to users authorized to play the associated role (2.14.2). So authorization for a specific action is granted to a user based on his expertise, on his present function and on the experiment status.

The AM has been implemented based on a client-server architecture (2.14.1): the clients send the authorization requests to the server that processes them against the access policies[15] and sends back the response to the clients. The RBAC data (users, roles, roles hierarchies, user assignment to role) are stored in an LDAP (Lightweight Directory Access Protocol) [115] database as this solution allows for easy interrogation from the access control software and offers the advantage of storing more information besides the user information.

Since autumn 2008 the AM has actively controlled the access to the TDAQ system. The use of the AM has been extended to perform access control also at the operating system level. It controls login on the nodes and access from the outside network to the ATCN and defines rules to allow specific users to execute particular privileged commands on the nodes by means of the "sudo" command.

Near the end of 2011 ~3800 user accounts and ~440 roles were defined and handled by an AM system consisting of 7 servers, one functioning as backup server. An example of the load (i.e. number of authorization requests per second) of one server, able to manage ~800 requests per second, is shown in figure 34.

Resource Manager. The Resource Manager controls the usage of the available hardware and software resources, which are described in the configuration database. Most of the resources are allocated or deallocated automatically when an application is launched or terminated via the Process Management service. If resources requested by a process are not available, the service denies to start the process. A GUI application is available which allows resources to be browsed and manually handled.

Process Manager. The Process Manager (PMG) [116] performs basic process control on behalf of the software components of the DAQ system. It is able to create and destroy software components distributed on the DAQ nodes, and to monitor their basic status (e.g. running, exited, killed). The Process Manager architecture consists of three main components (figure 35): the Process Manager Client, the Process Manager Server and the Process Manager Launcher.

The Client resides on the host where Process Manager requests are initiated and offers the user level interface to the Process Manager system. This interface provides tools to create and

[15] A policy is a set of rules applying to the resources specifying for each action associated with the resource the conditions for which permission is granted to execute the action.

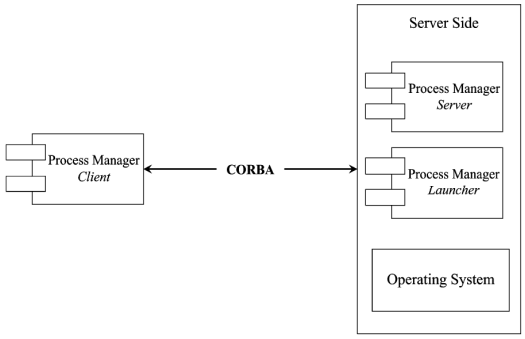


Figure 35. Architecture of the Process Manager Service. The server communicates with the clients using CORBA (2.2.1).

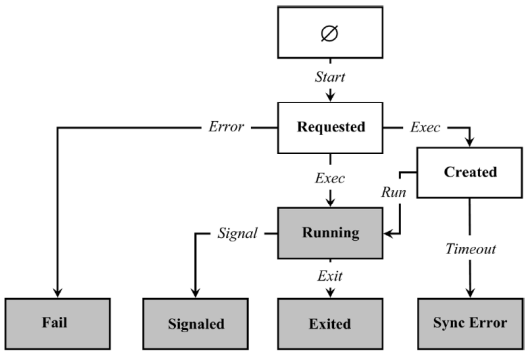


Figure 36. State machine of a PMG managed process. The states with a grey background are reported to the client.

manipulate processes in the whole system. One instance of the Server runs on each host where processes need to be managed. The Server acts as an information hub, dispatches requests and stores important data structures. It manages the process hierarchy, handles the resource allocation using the Resource Manager, handles the user authorization using the Access Manager and interacts with Launchers and Clients. Because of its complexity provisions are taken to ascertain that the Process Manager Server can restart in case of a crash. This implies storage of state information in the file system. Each process managed by the Process Manager system is handled by a Launcher. The goal of the Launcher is to have one single component that handles low level process management. The Launcher is owned by root and has the setuid attribute[16].

A process is represented by the Process Manager as a state machine, see figure 36. When a client application requests the start of a process the process goes into the REQUESTED state. After the process is properly launched it goes into the RUNNING state, otherwise it goes into the FAIL state. Processes configured to notify the Process Manager system when they are actively running go into the CREATED state. Once they confirm they are running, they go into the RUNNING state. If they fail to do so within a certain amount of time, they are first terminated and then go into the SYNC ERROR state. A running process can normally terminate (e.g. call the $exit()$ function) and go to the EXITED state. When a process terminates because of a signal (either an internal one, like segmentation fault, or an external one) it goes into the SIGNALED state. The Process Manager can terminate a process both by sending signals that the process can ignore and by sending signals that cannot be masked (e.g. $KILL$). It is also possible to combine the former two actions, i.e. first to send a TERM signal and wait for a defined amount of time for the process to exit, and then eventually to send a KILL signal. All states on the bottom line (FAIL, SIGNALED, EXITED and SYNC ERROR) are states that a process cannot be recovered from.

The Process Manager is able to tolerate the crash of certain elements of the system. If the Server crashes, it can be restarted. Upon restart, it reads information about running processes in the host file system and resumes processing. Clients crashing have a small impact on the overall system. The only vulnerable element is the Launcher and for this reason it is designed to be small and simple

[16]In UNIX systems the owner of a process is set to the owner of the file containing the executable if the file has the setuid attribute.

to minimize the probability of failures. The Process Manager has a small system resource usage. In real life operations the average CPU utilization is less than 15% when launching processes (on a machine with two Intel Xeon 5150 2.66 GHz CPUs running Scientific Linux CERN 4.5) and is virtually zero during the running period. The average memory usage is around 10 MB for the Server and less than 4 MB for the Launcher.

2.11.3 Core services: configuration

The Configuration service [117] provides access to the overall configuration of the DAQ system and to a partial description of the trigger and of the ATLAS detectors, which is stored in the Configuration Database. This database contains all information, from command line parameters to a specification of the overall organization of both hardware and software, needed for running the DAQ software. The service is provided by several components, which are described in more detail in the next sections.

DAQ, trigger and detector groups contribute their own part of the configuration description and implement the code for configuration of the applications using common configuration service tools. The description is based on the core object schema (a graphical representation of a part of it is shown in figure 37a) and can be extended by every group for their needs.

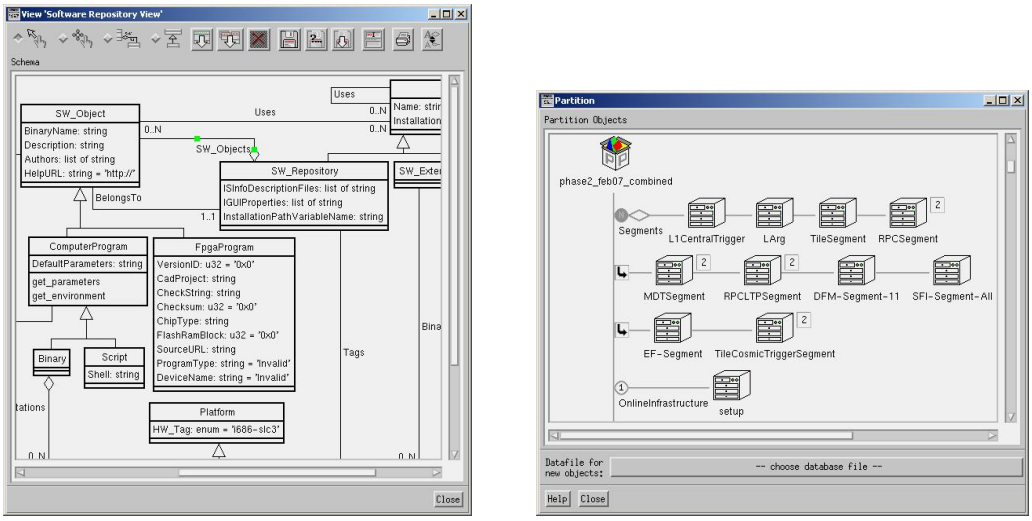
The configuration description is accessed simultaneously by every online application during the boot and configuration phases of a run. The configuration service notifies and reconfigures the clients (TDAQ applications) if any change of the configuration occurs during the run and archives the configurations used so that they later can be browsed by experts and accessed by applications performing offline event processing.

The configuration service consists of several components. The configuration description and object persistency is based on the in-house developed package OKS ("Object Kernel Support") (2.11.3.1). A Remote Database service (RDB) allows OKS objects to be accessed in a distributed environment. The application programming interface includes various Data Access Libraries (DAL) accessing OKS objects and implementing user algorithms on top of configuration data (2.11.3.2). The Run Number service (2.11.3.3) is used to assign a unique number for every data-taking run. The Resources Information package (2.11.3.4) handles the hardware and software resources (such as modules, channels, connections, etc.) that can be enabled or disabled during the data-taking run.

2.11.3.1 Storage and management of configuration data. OKS [118] is a set of tools providing object persistence. It is based on the object data model and supports named objects described by classes. For each class attributes (primitive types), relationships (links to other objects) and methods (actions on the properties of the object) can be specified. Multiple inheritance and polymorphism are possible. The integrity constraints allow restrictions on types and values of object's attributes and relationships to be defined. The OKS classes and objects can be created and modified dynamically, put into persistent storage and read back. A query language is available for effective data selection.

The main persistent storage for OKS is provided by XML files. The possibility to easily browse, modify and distribute these human readable files is one of the attractive points of OKS. Users can edit the schema (specifying classes) and data (specifying objects), in dedicated GUI applications (figure 37). However direct write access to OKS files is not allowed in the TDAQ system. To control the modifications of the database, a special mechanism (OKS Server) is deployed: before

2016 JINST 11 P06008



(a) The OKS schema editor showing a fragment of a configuration schema.

(b) The OKS data editor showing a fragment of the TDAQ readout configuration.

Figure 37. OKS schema and data editors.

committing a file into the ATLAS repository, the service checks the validity of the file for the ATLAS partition (the entire TDAQ system) and checks user permissions using the Access Manager. The database files are archived in a CVS [119] repository for future referencing and browsing of the history of changes, making use of the CORAL (2.2.4) interface. Incremental versioning is supported so that only differences between a complete base version of a database and its last revision are stored.

OKS provides remote access to the configuration information via the RDB service, making use of CORBA (2.2.1), (4.7.4). An RDB server provides its clients access to a single database, where the clients can be running on computers without a common file system. To address scalability issues, RDB servers can be organized in a tree-like structure: the master RDB server reads the XML repository and proxy RDB servers receive configuration information from the master. In the ATLAS TDAQ system, there is one RDB server per HLT rack. For efficiency reasons each RDB server keeps results of requests in a cache.

2.11.3.2 Configuration data access. The configuration data are accessed by applications implemented by developers from different groups. To ease software development the low-level algorithms for accessing the database data are completely hidden behind the high-level API of the DAL (Data Access Libraries) layer.

The DAL layer uses the lower abstract config layer to map the OKS database schema to C++, Java and Python classes and to instantiate objects according to the class-definitions. A DAL can be automatically generated from the database schema (by the "genconfig" tool), with it user-defined algorithms facilitating and simplifying complex or typical database queries can be developed.

The config layer provides an abstract interface for working with databases and for accessing configuration objects from different programming languages. The implementations of this interface are available as plugins for handling OKS XML files and OKS relational archives and for communication with the RDB server. This layer is used to work with an arbitrary database schema and

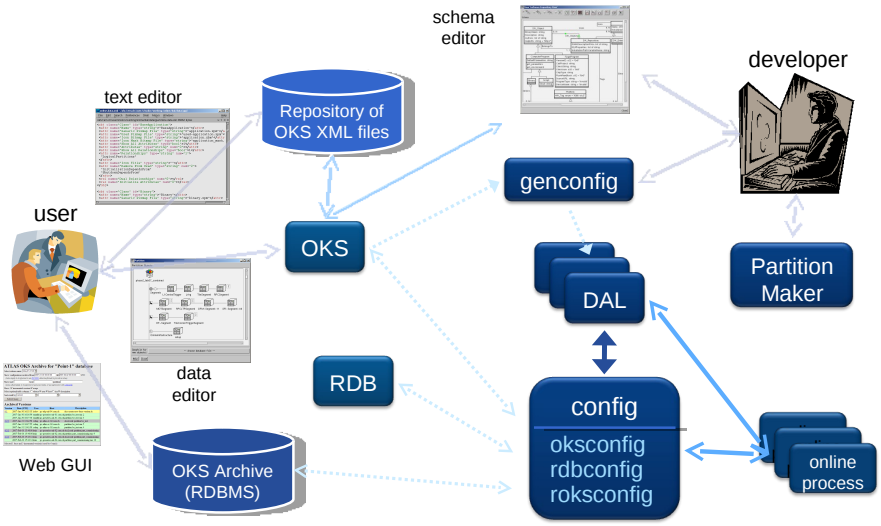


Figure 38. Configuration Databases: architecture, management and access.

data, as required by ATLAS control and infrastructure applications with user classes not known at compilation time.

Figure 38 shows the general architecture of the Configuration Database services and applications, and their use by TDAQ users and experts.

2.11.3.3 Run number service. The run number service allocates a unique number for every data-taking run and stores information about run parameters such as start time, duration and configuration, the Web interface allows the run number database to be browsed. For the implementation of the run number service use has been made of CORAL (2.2.4) for interfacing to an Oracle database.

2.11.3.4 Resource Information Service. A TDAQ resource is a part of the ATLAS TDAQ system that can be individually disabled (i.e. masked out from the configuration) or enabled without stopping the data-taking process. Resources can be interconnected, such that changing the status of a single resource may change the status of other resources. Information on whether resources are or were enabled or disabled needs to be available during data-taking as well as during offline data analysis.

Statically disabled resources are described in the configuration database. In case of problems during a run the Run Control stop-less recovery mechanism can disable some resources and inform the Resource Information service. This service maintains a list of enabled resources in the Information Service for online access. For offline access it also stores the list in the conditions database (the interface has been implemented using the COOL framework [42]). Experts can browse the resources archive.

PartitionMaker. The OKS editor can be used to create the XML files containing the configuration data needed to operate the TDAQ system. However, as entries have to be entered by hand it is a very tedious and error prone task to define large systems where a large number of objects (applications, configuration parameters, computers etc.) need to be described. For instance, a typical setup of the

2016 JINST 11 P06008

High-Level Trigger farms contains about 1500 multi-core nodes and over 13000 applications running on them. To facilitate reliable creation of the OKS files needed for large setups like this the Python tool "PartitionMaker" was developed. The tool allows the details to be specified in an algorithmic way. It describes the system configuration in terms of Python objects directly derived from the database schema, keeping the names of classes, attributes and relationships identical to the schema. In combination with the support by OKS for a "templated" description of a set of applications a compact description of very large partitions is possible. For instance, the layout of the L2 or EF farm can be specified as a group of "racks", each comprising about 30 nodes with 8 cores each.

For testing, PartitionMaker is often used to generate complete partitions. It can however also generate "segments" (2.1.2), which are meant to be included in other, often hand crafted partitions. This is most useful for segments that contain large repetitive patterns of objects. The HLT segments for L2 and EF that are used in the ATLAS data taking partition are generated in this way.

2.11.4 Expert system framework

The rule-based CLIPS expert system framework [120] was chosen as the core for the Run Control, Diagnostic and Verification System (DVS) and Online Recovery components presented in the following sections. CLIPS supports the storage of expert knowledge (in the form of if-then rules stored in human-readable text files) on testing sequences, recovery procedures and automation of different routine actions that can be performed by controls components on behalf of a person in charge of TDAQ operations. This knowledge base is complemented by a forward-chaining inference engine. CLIPS also supports procedural and object-oriented programming paradigms. These have been used to build a class hierarchy representing the applications and hardware in the system similar to the configuration schema of OKS. The corresponding objects are created dynamically using the information from the configuration database.

2.11.5 Run Control

Run Control (RC) is the distributed core framework [121] of the TDAQ Controls system. It allows the behavior of applications to be controlled according to a predefined state machine skeleton (shown in figure 39) by means of controllers joined in a hierarchical distributed tree structure. The latter guarantees synchronous and homogeneous execution of commands through the whole system. A transition is only possible after all controllers in the tree have completed the previous transition and thus have reached the same state. The top-level Run Controller (called the Root Controller) reports the status of the whole TDAQ system and accepts commands from the person in charge of managing Run Control. Leaf controllers deal with applications that directly control the TDAQ hardware or perform data-taking or trigger actions according to the state machine state and transitions. The intermediate controllers represent states of different segments (2.1.2) of the TDAQ system. The generic structure of the RC tree is shown in figure 40. Actions associated with the transitions are specified in table 5. The Run Controller also contains a local recovery unit, based on an expert system (2.11.7). Table 6 contains an overview of the RC infrastructure for the full ATLAS TDAQ system.

The overhead introduced by the RC framework command distribution and for Finite State Machine (FSM) transitions has been studied. Special test TDAQ partitions with "empty" applications were used, therefore the time that would be spent in transitions by subsystems in a real data-taking partition is not included in the results. Tests have been done with a controller managing the 30

2016 JINST 11 P06008

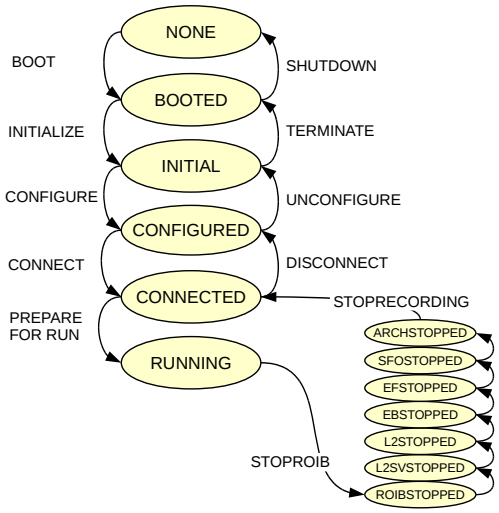


Figure 39. Run Control Finite State Machine. See table 5 for the actions associated with the transitions.

Table 5. Run Control transitions and actions (infrastructure applications, e.g. IS and MRS servers, provide services for other applications).

Transition	Actions performed by controllers and applications
BOOT	Test of hardware, start of child controllers and infrastructure applications.
SHUTDOWN	Infrastructure applications and child controllers terminated.
INITIALIZE	Start of data flow applications.
TERMINATE	Termination of data flow applications.
CONFIGURE	Applications (re)read configuration from database and perform internal configuration.
UNCONFIGURE	Clear internal application structures.
CONNECT	Establish connections to data flow applications.
DISCONNECT	Terminate connections to data flow applications.
PREPARE FOR RUN	Applications read conditions data and get ready for events.
STOPROIB STOPL2SV STOPL2 STOPEB STOPEF STOPSFO STOPARCHIVING STOPRECORDING	A set of stop transitions to clear sequentially all buffers in the data flow - trigger chain.

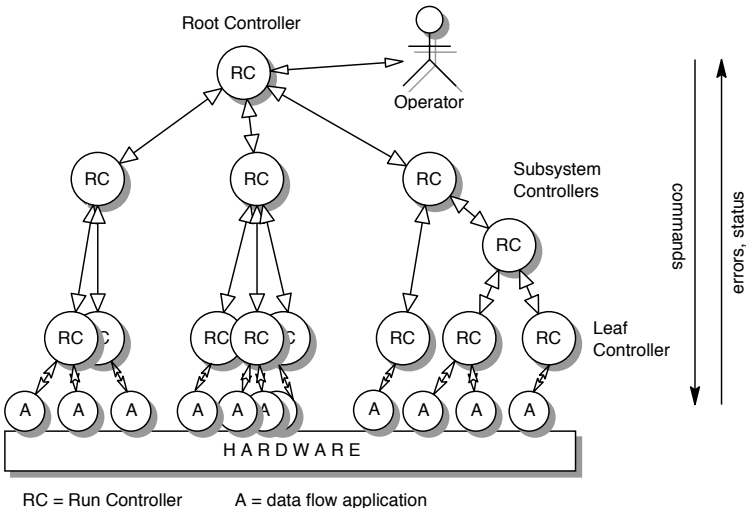
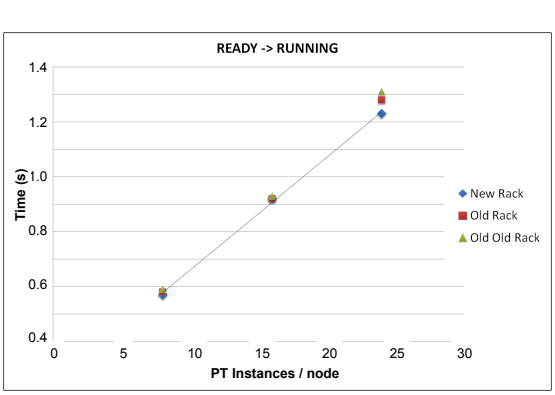


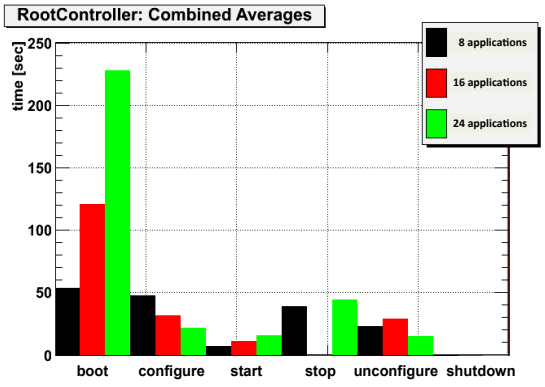
Figure 40. Run Control Tree.

Table 6. Run Control infrastructure (end 2012).

Controlled hosts (PMG agents)	2300
Controllers	220
Infrastructure applications	1450
Data flow and trigger applications	33000



(a) Overhead of the RC command distribution in a single transition.



(b) FSM transition times for an RC tree controlling 30 nodes with 8, 16 or 24 applications.

Figure 41. Run Control framework performance.

2016 JINST 11 P06008

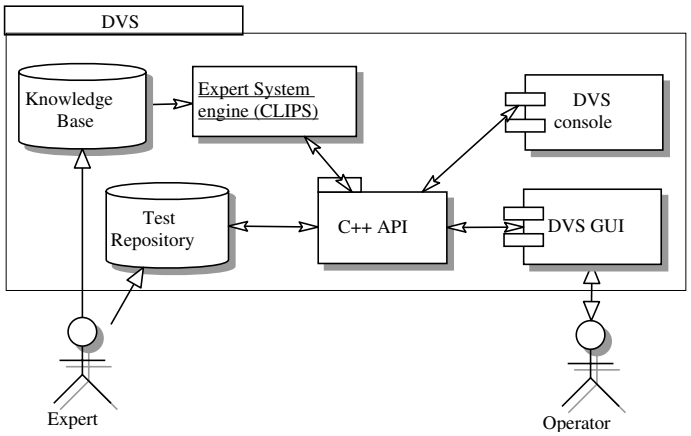


Figure 42. Architecture of DVS.

HLT nodes of a rack and 8, 16 or 24 applications per node. Figure 41a shows the time taken by the Root Controller for a single FSM transition from the most simple transition from the READY state (available in an earlier version of the RC FSM) to the RUNNING state as a function of the number of controlled applications. This is the pure overhead time of the RC framework command distribution. Figure 41b shows the total time taken by the Root Controller (which is the aggregate time taken by transitions in the whole tree) for different FSM transitions, including time to launch the applications in BOOT and to read configuration data in CONFIGURE. The increase of the time needed for the BOOT transition for 16 and 24 applications per node relative to that needed for 8 applications per node can be attributed to simultaneous starting of the applications, which creates an excessive load. This issue has been addressed in an update of the RC framework by starting the applications in a more random manner on different nodes in the rack.

2.11.6 Diagnostic, testing and verification framework

The Diagnostic and Verification System (DVS) [122] is based on the CLIPS expert system framework (2.11.4). It allows a user to automate testing of complex TDAQ configurations and to diagnose problems found. The architecture of the system is shown in figure 42. It combines the information about available tests from the Test Repository (an OKS database which is a part of the partition configuration), diagnostics rules from its Knowledge Base and information about the current configuration of TDAQ and makes it possible to automate testing of the configuration using a GUI or a command-line application. It can diagnose problems detected during testing, provided that relevant knowledge is stored in the Knowledge Base. Subsystem experts can add subsystem-specific tests to the Test Repository and custom rules to the Knowledge Base.

The Test Repository database schema [123] allows a test (or a number of tests) for any component in the system to be defined and configured. The test is a small application which is launched via the Process Manager service and which verifies the functionality of a component. This functionality is used within the RC framework in two different ways: 1) in the initialization phase, all hardware components are tested before launching any applications on them; 2) in case of a failure, a subset of the system is tested to help localize and diagnose the problem.

2016 JINST 11 P06008

The Test Repository database contains a few dozen tests for basic TDAQ hardware (e.g. computers, NIC cards, ROBIN cards) and software components. During start up of the full ATLAS TDAQ system, in total $O(10^4)$ test processes are executed in a few seconds. To meet the performance requirements, the load associated with management of the test processes is shared by a cluster of 15-20 nodes.

2.11.7 Online recovery and error handling

Online error recovery system. There are two main parts of the dynamic recovery system [124]: a local unit and a global unit. The local unit is integrated with each controller in the control tree. It handles the situations that can be dealt with locally at a segment level (e.g. in an HLT node), that is, issues that do not have an immediate effect on the rest of the system. It can set the error state of the controller, but is also able to perform more advanced actions such as restarting applications or notifying other applications, or temporarily holding the trigger to perform a recovery action. All errors gathered by the local unit are reported to the global unit, including information such as whether the action has been taken and whether the problem in question has been solved or not. The global unit handles situations that have a system-wide impact. As an example, the global unit deals with the selection of the clock for the overall experiment, which varies depending on the status of the LHC.

One of the main tasks of the online recovery system is to maximize data-taking efficiency and data quality. To do this it reacts to operational information provided by the Information Service or messages sent by the Message Reporting System. If any of the RODs is not working correctly and has asserted a permanent BUSY to the trigger, the expert system is able to mask that part of the readout, such that data taking will continue without it (this is known as stopless removal). If a part of a sub-system looses synchronization with respect to signals used for event identification (e.g. the ECR), the expert system will temporarily hold the trigger, provide the correct information to the system and allow it to recover before resuming data taking. Finally, if a part of the readout that had been previously disabled in the course of the same run can be enabled again, the expert system is capable of re-including it into the ongoing run by temporarily holding the trigger and synchronizing the recovered part with the rest of ATLAS (this is known as stopless recovery). Whenever possible, the actions of the online recovery system do not introduce dead time. In table 7 actions are listed for which it is necessary to cause an interruption of data taking together with typical recovery times.

2.11.8 Integrated Graphical User Interface

An operator in charge of data taking can interact with the TDAQ system via the Integrated Graphical User Interface (IGUI). The tasks of the IGUI can be coarsely grouped into three categories:

- system status monitoring: presentation of the global status of the TDAQ system and of the ATLAS run, as well as reporting of errors and output of other messages generated by the system,
- control: interact with the TDAQ Run Control (2.11.5) and Online Recovery system (2.11.7),
- configuration: presentation and modification of the configuration of the TDAQ system.

The IGUI is implemented in Java (version 1.6) in view of the versatility and completeness of the Swing GUI components [125] and the availability of third-party libraries providing several

2016 JINST 11 P06008

Table 7. Online recovery actions and the length of the resulting interruption to data taking.

Action	Time	Reason
Change of clock	2 s	After a clock switch some of the detectors need some time to adjust to the new clock (not critical as it never occurs during stable beams).
Holding trigger during LHC ramp	minutes	While LHC ramps the energy of the beams, the clock frequency changes. Some of the detectors cannot take data in these conditions. Triggers are resumed when the beam energy reaches 2.7 TeV as all detectors can sustain the clock variations from then on.
Stopless removal	2 s - minutes	The dead time is not provoked by the recovery system, but by the faulty component. The time needed for the detection of a faulty ROD is sub-detector dependent and is in the range of 2–60 seconds. Nevertheless, the expert system usually is configured to ask the operator for confirmation before excluding a part of the readout, thus this operation will take potentially much longer.
Resynchronization	2–8 s	The trigger is held during resynchronization to ensure that the ECR count does not change while it is being downloaded into the desynchronized components (RODs). The time to resynchronize is detector dependent.
Stopless recovery	2–10 s	The actions to be taken are very similar to those required for resynchronization.
Dynamic removal of an L2SV	~10 s	The RoIB needs to empty all its input and output buffers and to reconfigure itself without the faulty L2SV.
Restart of all or part of the TTC partitions associated with a sub-detector	~10 s - minutes	A set of actions initiated by the operator, aimed at resolving problems with the readout of a sub-detector and consisting of holding the trigger, reconfiguring and restarting the faulty part of the readout and starting the trigger.

additional graphical widgets. The core part of the IGUI main window, shown in figure 43, contains the main interfaces to the Run Control, the Message Reporting System and a tabbed pane with run information. The remainder of the window consists of a modular part consisting of a tabbed pane which can host panels covering additional functionality and of buttons for opening additional windows for interacting with several services, e.g. the Information Service (IS).

The load of the IGUI is high during Run Control FSM transitions: all the run controllers and state-aware DAQ applications (in total $O(10^4)$ update their state (globally up to $O(10^5)$ updates per second) and many MRS messages are being sent (with peaks of 1000 messages per second). The IGUI has been shown to be capable of sustaining this load keeping its responsiveness and, at the same time, promptly updating the status of the system.

2.11.9 Shifter Assistant

During data-taking runs, streams of messages sent by DAQ and detector applications via the MRS system together with data published from applications via IS are the main sources of information on the running operations. The huge flow of messages produced (with an average rate of the order of

2016 JINST 11 P06008

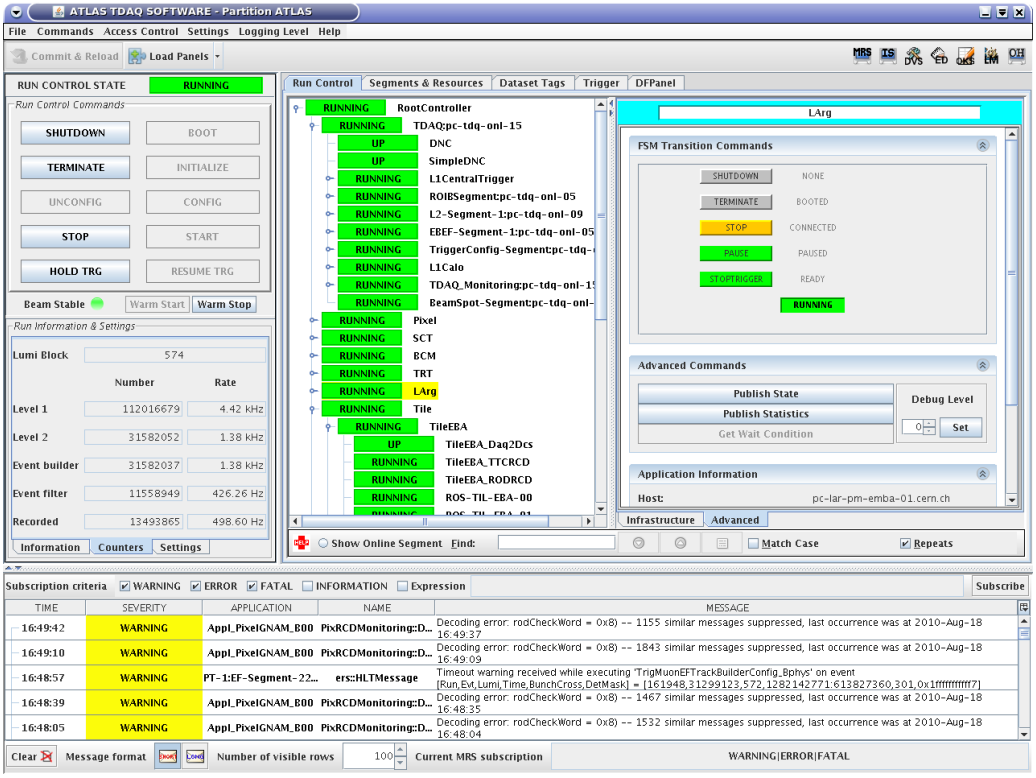


Figure 43. IGUI main window with the ATLAS partition taking data and RC in the RUNNING state.

1–10 kHz) is constantly monitored by DAQ operators and experts to detect problems or misbehavior of the system.

The Shifter Assistant application [126] is meant to reduce the man power needs and to ensure a constant high quality of problem detection by automating most of the monitoring tasks and providing real-time correlation of data-taking and system metrics. The application is composed of the following main components: an interface to the main sources of TDAQ operational data (MRS, IS, RDB, LHC via the Detector Control System (DCS) [127] and the Nagios monitoring system (2.14.2) [128]), a core processing engine (a Java implementation of the Complex Event Processing engine [129]) responsible for correlation of events through expert-defined queries, and a web based front-end to present real-time information and interact with the system.

The Shifter Assistant is able to extract, aggregate and correlate relevant information from the flow of operational data and to provide real-time feedback in form of alerts to DAQ operators and experts who can promptly react when needed. The Knowledge Base includes a few dozens of rules which cover most common operational problems and also regular system checks.

2.11.10 Auxiliary applications for control

Analytics dashboard. During data-taking runs, the streams of messages sent by applications are constantly monitored by experts for any problems. The TDAQ Analytics Dashboard [130] is a web application that collects, correlates and visualizes this real-time flow of information to simplify and improve system analysis and error detection tasks. It is composed of a correlation engine

2016 JINST 11 P06008

that performs aggregation, processing and filtering of real-time streams of messages, and of web applications to visualize the data collected.

Control Room Desktop. The Control Room Desktop (CRD) is the GUI environment available on the desktop machines in the ATLAS Control Room. It is based on the Linux K Desktop Environment (KDE) [131] and exploits its KIOSK [132] configuration mode. This allows the capabilities of a KDE environment to be defined and restricted based on the credentials of the users. When the user logs into a control room machine, the access control of the CRD retrieves the user role data from the Access Management system (2.11.2) and decides which KIOSK profile to load for the user. The GUI environment offered to the user is generated from the KIOSK profile so the user has access only to the functions and applications specific to his current roles.

Data taking efficiency tools. The databases with information about LHC beam status, run start and stop times, Ready For Physics and trigger Busy flags are queried by tools that compute statistical information on DAQ efficiency (efficiency per fill, fill duration, Ready For Physics flag delay, inefficiency sources etc.) and that present the results in graphical form on web pages.

Electronic logbook. A web-based electronic logbook allows users to store and retrieve information associated with different types of operational events. For the start or stop of a run for example run parameters, run duration, number of events and shifter comments are recorded. The tool also provides an API for insertion of information and allows e-mail notification for certain types of events.

Archiving and browsing of log files. The large volume of log files produced by the TDAQ applications are initially stored on local disks. These files are accessible online via a web browser. An archiving service is run regularly to clear local disk space and archive all the log files for possible later offline access.

2.12 Monitoring infrastructure

The ATLAS monitoring system is organized as a distributed framework. It includes core software services for information-sharing, dedicated monitoring facilities and graphical monitoring displays (see figure 44). The framework components and services can be easily adapted to the requirements of different ATLAS sub-detectors and to monitoring requirements at different levels of the data-flow chain. The ATLAS TDAQ and sub-detector systems are monitored in two different ways:

- Operational monitoring: operational data and functional parameters of the hardware and software components are collected and published to the monitoring applications.
- Event data monitoring: results of analysis of sampled events as well as the HLT algorithms online monitoring information are provided to the monitoring applications.

2.12.1 Core services

The online monitoring services are built on top of common software components for inter-process communication (the IPC service, see section 2.2.1) and for information sharing (the IS service, see section 2.2.2). The following services can be distinguished:

2016 JINST 11 P06008

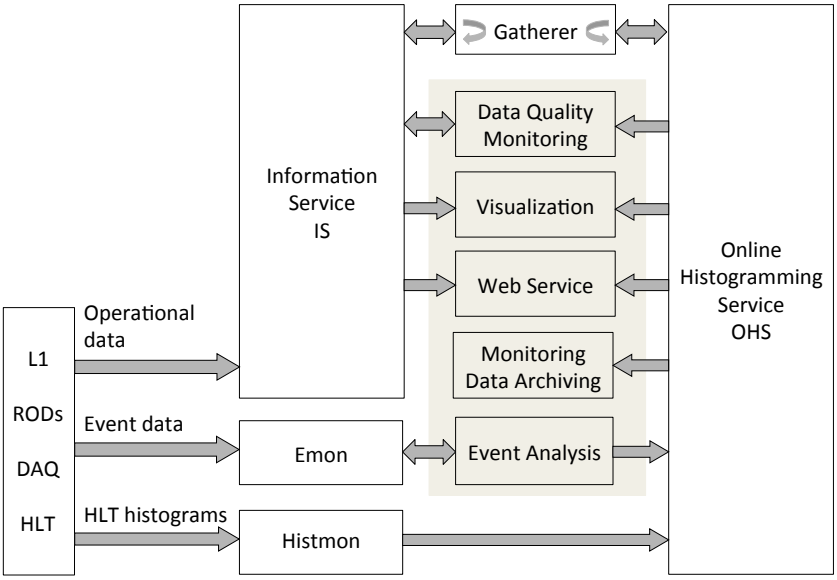


Figure 44. Monitoring infrastructure, consisting of core services (OHS, Emon, Histmon and the Gatherer), frameworks for analysis of raw event data and of histograms, tools for archiving and visualization and a service providing remote access to monitoring results.

- **Online Histogramming Service (OHS)** based on the IS service. It handles histogram objects and provides transient storage between producers of histograms and applications displaying histograms. Histograms are published to OHS and are made available to the entire system.

- **Histmon** for updating and publishing histograms periodically. Once a histogram is registered with Histmon, the package cares about publication and update of the registered histograms. Histograms can be registered in different categories with freely configurable update intervals. Updates may be performed every few minutes for high priority histograms to only once per run for histograms with low priority. The HLT publishes histograms in Histmon via a special version of the Athena histogram service THistSvc. This special version makes use of the abstract Athena interface ITHistSvc, which defines the histogram service interface to Athena algorithms, and interfaces also to the Histmon package and its registry. The corresponding offline Athena service, which makes use of the same interface, saves the histograms in regular ROOT [133] files.

- **Event monitoring (emon)** for event sampling and distribution of event data. Users may request event fragments with selected properties such as trigger type, from a specific sampling point. Emon does not depend on the event format and can handle events coming from any level of the data flow.

- **Gatherer** [134], which combines monitoring results from various providers and retransmits the combined data back to IS or OHS, depending on whether the supplier of the data is IS or OHS. The output of the Gatherer mainly consists of sums or averages of histograms, but can also include sums of trigger rates and more complicated objects. To optimize the performance of the histogram collection mechanism the Gatherer applications in L2 and EF

2016 JINST 11 P06008

are organized in a tree like structure with one Gathering application per HLT rack. These publish their results to top level Gatherers for the L2 and EF farms respectively. A rack level Gatherer typically needs to process about 750000 histograms within an update interval of 2 minutes. Internally each Gatherer therefore processes the histograms in parallel and sums e.g. histograms that do not share resources in concurrent worker threads.

2.12.2 Monitoring framework components

The monitoring framework consists of the following components:

- **GNAM** [135] is a light-weight configurable framework used for event-content analysis and histogram production, optimized for detector monitoring. Plugins with detector-specific code, typically for event decoding, data analysis and histogramming, are loaded at runtime. The application core interacts with Run Control, the configuration service and the histogramming and event monitoring services. It also provides support for custom data exchange between the loaded plugins. This capability can for example be exploited to build histogramming libraries correlating data of different sub-detectors or to feed multiple histogramming plugins with a single decoding step.

- The **data quality monitoring framework** [136] allows automatic analysis of monitoring histograms with statistics algorithms according to user-defined configurations. A summary of the results produced by the algorithms is published to IS and is also archived for future offline use. The data quality monitoring display (2.12.3) allows the analysis results to be visualized. The framework consists of a core part, which executes the algorithms and communicates via plugins with the configuration database, the conditions database and IS. The framework provides already a number of predefined algorithms for common operations on histograms like comparison or fitting. With custom plugins different input sources and output destinations can be selected and the framework can be extended with new algorithms.

- **Monitoring data archiving:** ATLAS produces up to 10 GB of monitoring data per run. Most of this data need to be available offline so that offline analysis results can be cross-checked with online monitoring results. The monitoring data archiving tool takes care of long-term storage of these results [137]. The data is transferred to the CERN mass storage facility and their storage location is registered in the conditions database. The tool provides a local data cache to hold monitoring data after the end of run, making fast access to the histograms of the last runs possible for a limited time span.

2.12.3 Visualization tools

A series of tools is available to help assess the correct functionality of the system visually:

- The **IS monitor** can be used to visualize the content of all IS servers while the system is in use.

- The **data quality monitoring display** [138] is a high-level graphical user interface that presents results produced by the data quality monitoring framework (2.12.2). In a summary panel the overall status is available with color-coded status entries per sub-system: red signals an error state, yellow indicates a warning, while green reflects a good status. These results

2016 JINST 11 P06008

and the monitoring histograms are presented in a tree hierarchy. The user can "zoom in" to individual systems to investigate specific problems. A history panel gives the possibility to follow the time evolution of the data quality results. A graphical representation of the sub-systems and their components using detector-like pictorial views is possible. These views can be created with a specialized tool developed for this purpose.

- The **online histogram presenter** [139] handles and displays histograms published to OHS (2.12.1). The architecture has been optimized with respect to minimization of resource utilization in terms of network traffic, CPU utilization and memory usage. A set of plugins provides simple operations such as browsing of all available histograms and ordering and updating of histograms in the GUI tabs. Commands can be sent to monitoring applications by means of the OHS message service. Custom plugins can be developed for browsing, retrieval and notification of histogram updates. The GUI is based on the Qt framework [140] and embeds ROOT [133] for histogram visualization and graphical handling. The application can be extensively customized by means of configuration files, e.g. histograms can be displayed using different drawing options or they can be displayed with overlaid reference histograms.

- The **trigger presenter** [141] calculates, displays, monitors and archives any quantity that varies with time, in particular trigger rates. It is implemented as a suite of distributed applications (figure 45). So-called adapters calculate the rates, format the results and then publish them to a specific IS server (2.2.2). The trigger rates are kept in memory on this IS server for about 3 hours. HLT rates, which have to be determined for each trigger chain with a relatively small integration time of about 10 s, are measured individually for each of the several thousand HLT processes. The results are summed by the adapters or the sums can be received directly from the Gatherer. Additional adapters are responsible for measuring the L1 rates per trigger item, and per trigger sector. Separate adapters take into account the luminosity and pile-up measurements available from the IS Server to compute the predicted L1 and HLT rates that serve as a reference to the online measured rates. Other applications can subscribe to the trigger rate specific IS server and get the published rates. The GUI of the trigger presenter is based on the Qt [140] framework and displays the time dependent quantities in tables and in time series plots. A WMI plugin (2.12.4) builds and publishes web pages with trigger rate information for the last 24 hours on a public web server. The web pages are saved every day and the rates are archived in a ROOT file with a fine grained time resolution of about 10 s. Later these files are stored on the CERN mass storage facility Castor.

- The **operational monitoring display** reads IS information with respect to a given configuration. It calculates for selected IS quantities averages, sums and standard deviations and displays them as time-series, bar charts, distribution graphs or tables. Threshold checks can be applied to retrieved IS quantities. The quantities can be grouped together or be converted to histograms and published to OH servers. The display is configured by means of a point-and-click GUI and can be dynamically changed during runtime. The application can run in GUI-less mode as part of a partition and serve as a bridge between OHS based applications and IS information.

2016 JINST 11 P06008

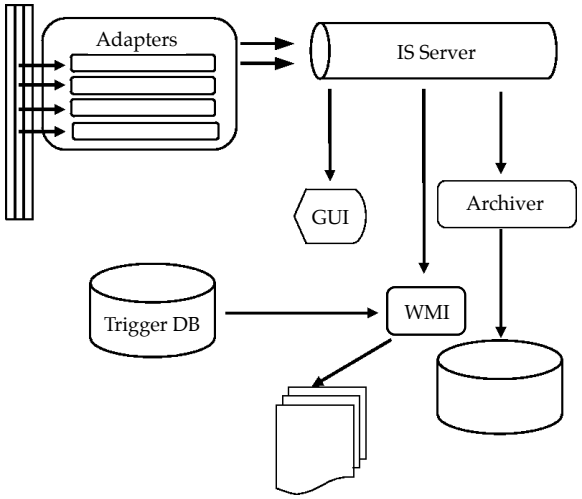


Figure 45. The parts of the trigger presenter package (adapters, GUI, archiver and a Web Monitoring Interface plugin) and the associated monitoring data streams.

- The **MonaIsa** tool also reads quantities from IS and produces monitoring histograms of several values of interest for each component of the data flow infrastructure, such as input and output rates, backpressure, etc. Each histogram corresponds to a single variable of interest, and each histogram bin corresponds to a single application, such that there is one bin per active data flow application (for example for a set of SFI applications there is one bin for each application in the set), and each bin is labeled appropriately with the name of it.

- The **event dump tool** is dedicated mostly to experts. It can display the content of a physics event that has been received from the event monitoring service (emon) or has been read from a file.

2.12.4 Remote monitoring

Remote access to monitoring information is an indispensable component of the experiment's operational model. This includes access to monitoring histograms, to the detector control state and to the data quality status, either via web-based services or via direct export of monitoring displays. Three types of service can be distinguished:

- **Static web monitoring** is supported by the Web Monitoring Interface (WMI) software service, which executes a number of plugins providing information on different sub-systems, converts this information into HTML pages and stores them. The pages contain a relatively small subset of the most important parameters representing the overall global status of the TDAQ system. There are four plugins publishing the overall state of the data-taking and trigger sub-systems as well as providing general information about data quality and run efficiency.

- **Dynamic web monitoring** provides a set of dynamically constructed web pages, which update information requested by a user in real-time by taking it from the TDAQ online services (e.g. Information Service and Configuration Service).

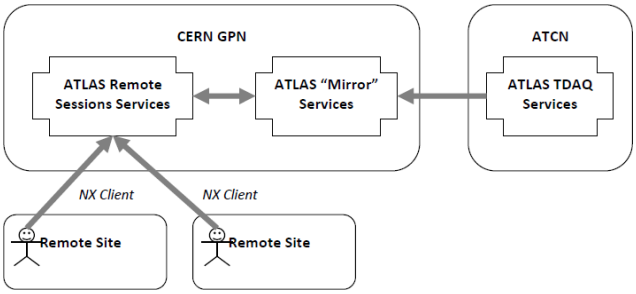


Figure 46. Remote monitoring system architecture.

- **Real-time "mirror" monitoring** makes a real-time copy of the monitoring and configuration information from the TDAQ services connected to the ATLAS Control Network (ATCN) to their mirror counterparts connected to the CERN Global Public Network (GPN). Access to this service is restricted to a limited number of users. The information is always passed one-way, from the ATCN to the GPN, to preserve the security of ATLAS data taking. Any network connections from the "Mirror" monitoring nodes to the nodes connected to the ATCN are prevented. The delay of information transfer is of the order of a few milliseconds. The standard monitoring GUI applications, the same as used in the main ATLAS Control Room, can be used, so there is no need for an extra training for remote users. Since those applications are running on dedicated machines located at CERN, an efficient and reactive way of passing screen information from CERN to remote sites is required. By using the NX technology [142] the problems with X11 display export for long distance connections are avoided, due to the on-the-fly compression and extensive caching provided by NX on top of the standard X11 protocol. The ATLAS remote monitoring system architecture is shown in figure 46.

2.13 HLT and data flow resource utilization assessment: cost monitoring

The HLT must analyze events within the limits imposed by the available computing power, network bandwidth, and storage space. To understand the resource utilization in the TDAQ data flow system and in the HLT the so-called "cost monitoring framework" [143] was introduced. It records performance data on an event by event basis and saves the data in a special data stream for further offline processing. The data are also collected for events that are rejected at a later stage and hence are inaccessible for offline analysis. The information provides valuable input for optimization of the HLT and of data flow, and for predicting data flow requirements associated with future trigger operation.

The performance data are collected by a dedicated monitoring tool. Global data, such as the run and event numbers, the total HLT processing time and a detailed record of the HLT trigger decision process, are stored for every event. To determine the performance of HLT algorithms, additional information is sampled for every 10th event. The steering framework is instrumented to record, for every HLT algorithm executed: its name, the trigger signature which requested the algorithm execution, the properties of the region of interest and the total algorithm execution time. As the input data for L2 needs to be retrieved via the data flow network, additional quantities are recorded for L2, like the time stamps for start and stop of every retrieval of ROB data over the network, the list of ROB identifiers from which data are requested and their data volume. Furthermore a record is kept of whether an algorithm request of ROB data could be served from the cache with ROB

2016 JINST 11 P06008

```
ROS:                          ROS-SCT-BA-00
Processed events:             7894
ROB DataCollector calls:      27678
  - retrieved:                9867
  - cached:                   12766
Size of requested ROBs:       78291.8 KB
  - retrieved:                34064.7 KB
```

Algorithm	ROS requests	ROS retrievals	Requested data (kB)	Retrieved data (kB)	Cached data (kB)	Request rate (Hz)	Retrieval rate (Hz)	Cache rate (Hz)	Requested bandwidth (kB/s)	Retrieved bandwidth (kB/s)	Cached bandwidth (kB/s)	Requests per call	Retrievals per call	ROBs per retrieval
TrigL2SiTrackFinder_MuonA	7103	2532	14710.23	6266.08	8444.15	16014.33	5708.61	7401.81	33165.49	14127.42	7401.81	1.18	0.42	2.2
TrigL2SiTrackFinder_TauB	4798	2200	9242.00	6876.09	2365.91	10817.51	4960.09	913.11	20836.89	15502.73	913.11	1.49	0.68	2.8
TrigIDSCAN_Bphysics	2423	1170	10044.43	4304.65	5739.78	5462.86	2637.87	2195.97	22646.05	9705.21	2195.97	1.02	0.49	3.3
TrigL2SiTrackFinder_eGammaA	2422	1025	4069.05	2567.07	1501.98	5460.61	2310.95	1203.95	9174.04	5787.69	1203.95	1.22	0.52	2.2
TrigL2SiTrackFinder_JetB	2710	959	7781.54	2997.14	4784.40	6109.93	2162.15	3027.91	17544.16	6757.31	3027.91	1.19	0.42	2.6
TrigL2SiTrackFinder_BphysicsB	2593	756	12708.36	2805.48	9902.87	5846.14	1704.47	4110.11	28652.09	6325.21	4110.11	1.00	0.29	3.2
TrigL2SiTrackFinder_MuonB	3710	658	9396.07	1688.23	7707.84	8364.52	1483.52	6881.00	21184.26	3806.25	6881.00	1.00	0.18	2.3
TrigL2SiTrackFinder_BeamSpotB	418	418	6190.67	6190.67	0.00	942.42	942.42	0.00	13957.41	13957.41	0.00	1.00	1.00	11.0
TrigL2SiTrackFinder_muonIsoB	201	136	534.63	354.92	179.71	453.17	306.62	146.55	1205.37	800.20	146.55	1.00	0.68	2.4
TrigIDSCAN_eGamma	101	7	263.09	8.85	254.24	227.71	15.78	211.93	593.17	19.96	211.93	1.00	0.07	1.1
TrigL2SiTrackFinder_TauC	21	6	56.31	5.53	50.79	47.35	13.53	33.82	126.96	12.46	33.82	1.00	0.29	1.0
TrigL2SiTrackFinder_MuonC	57	0	151.77	0.00	151.77	128.51	0.00	128.51	342.19	0.00	128.51	1.00	0.00	0.0
TrigSiTrack_Muon	52	0	134.56	0.00	134.56	117.24	0.00	117.24	303.37	0.00	117.24	1.00	0.00	0.0
TrigIDSCAN_Muon	51	0	132.34	0.00	132.34	114.98	0.00	114.98	298.37	0.00	114.98	1.00	0.00	0.0
TrigL2SiTrackFinder_TauIsoB	1	0	3.79	0.00	3.79	2.25	0.00	2.25	8.54	0.00	2.25	1.00	0.00	0.0
TrigL2SiTrackFinder_eGammaB	711	0	1890.03	0.00	1890.03	1603.01	0.00	1603.01	4261.23	0.00	1603.01	1.00	0.00	0.0
TrigL2SiTrackFinder_JetC	1	0	3.32	0.00	3.32	2.25	0.00	2.25	7.48	0.00	2.25	1.00	0.00	0.0
TrigL2SiTrackFinder_muonIsoA	1	0	0.93	0.00	0.93	2.25	0.00	2.25	2.09	0.00	2.25	1.00	0.00	0.0
TrigL2SiTrackFinder_TauA	21	0	71.49	0.00	71.49	47.35	0.00	47.35	161.18	0.00	47.35	1.00	0.00	0.0
TrigSiTrack_eGamma	106	0	290.47	0.00	290.47	238.99	0.00	238.99	654.90	0.00	238.99	1.00	0.00	0.0
TrigL2SiTrackFinder_JetA	1	0	3.32	0.00	3.32	2.25	0.00	2.25	7.48	0.00	2.25	1.00	0.00	0.0
TrigL2SiTrackFinder_BphysicsA	54	0	278.45	0.00	278.45	121.75	0.00	121.75	627.78	0.00	121.75	1.00	0.00	0.0
TrigL2SiTrackFinder_TauCoreB	1	0	3.79	0.00	3.79	2.25	0.00	2.25	8.54	0.00	2.25	1.00	0.00	0.0
TrigL2SiTrackFinder_eGammaC	121	0	331.18	0.00	331.18	272.81	0.00	272.81	746.67	0.00	272.81	1.00	0.00	0.0
Total	27678	9867	78291.8	34064.7	44227.1	62402.5	22246.0	28782.1	176515.7	76801.9	99713.8	N/A	N/A	N/A

Figure 47. Example output of the framework for analysis of cost monitoring data. Among other information the table shows the requesting algorithms, for the ROS PC considered the partial access rate of each algorithm of the cache ("request" rate) and of the ROS PC itself ("retrieval" rate), the amount of data transferred from the cache to the algorithm and from the ROS PC to the cache and the average number of ROBs for which data is requested from the cache. The events have been collected over a period of 15 minutes. The number of events is relatively small due to the nature of the sampling mechanism on the one hand and as only a fraction of the cost data collected is analyzed on the other hand.

data (2.1.3), (2.9.1) or triggered a network access. The cost monitoring records contain also the complete sequence of algorithm executions required to arrive at the HLT decision.

The data collected are stored in a circular buffer and a special monitoring trigger is used to read out the cost monitoring data. The complete buffer is attached to an HLT result only if the event is rejected by all physics signatures. With the partial event building mechanism (2.6.2) only the HLT record is saved, while all other detector data are discarded. In this way these transient monitoring data do not enter permanent tape storage reserved for physics data.

The monitoring data are copied to the Tier-0 processing centre, where they are stored in ROOT files. The ROOT files from a given ATLAS run are processed together to generate Web accessible summaries. Figure 47 shows an example in the form of a detailed summary table of the data access to a given ROS PC of the SCT. The table allows the algorithms which generate the highest load on the ROS to be spotted quickly. This information was used on the software side to optimize the physics selection menu and on the hardware side to assign read out links in an optimal way to the ROS PCs of the pixel detector.

The same monitoring framework and analysis code is furthermore used for offline studies of the trigger performance. One application is the extrapolation of trigger rates to higher luminosities, which requires knowledge of event by event trigger decisions. The "cost monitoring" framework allows the trigger rates for the three levels of the trigger system to be predicted within a few minutes, a capability that was of critical importance for the development of trigger menus during the luminosity ramp up phase of LHC in 2010.

2016 JINST 11 P06008

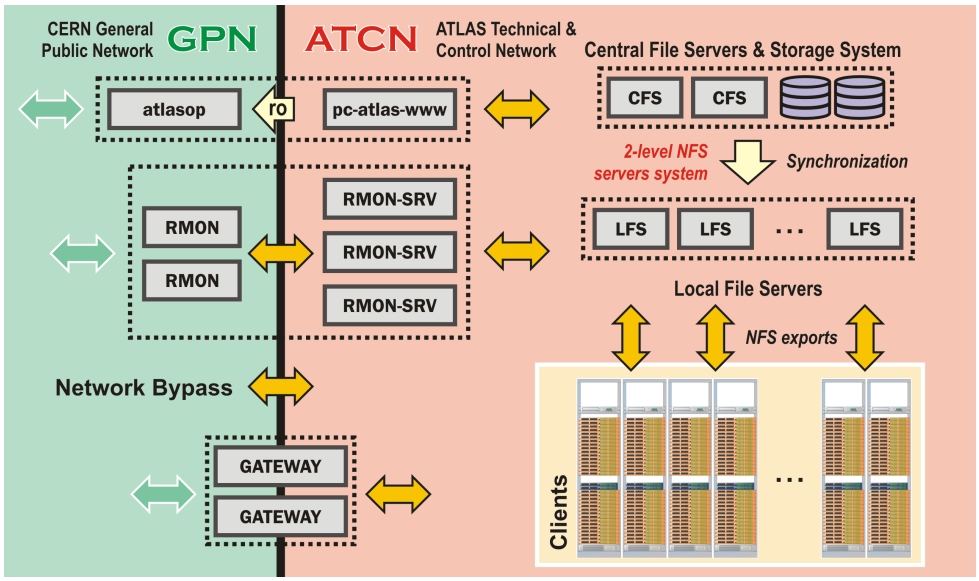


Figure 48. ATLAS DAQ/HLT IT infrastructure: RMON refers to remote monitoring nodes while RMON-SRV to servers associated with these nodes; LFS and CFS refer respectively to Local and Central File Servers; atlasop and pc-atlas-www are web servers.

2.14 System administration

2.14.1 DAQ/HLT computing infrastructure

The trigger and data acquisition infrastructure contains ~ 2000 nodes. About 200 are situated underground in the USA15 service cavern, while the remaining are installed in almost 100 racks on the surface, inside a barrack in the SDX1 building (2.18.2). The computing infrastructure also includes approximately 100 machines located near the main control room and in various satellite control rooms.

The system infrastructure is depicted schematically in figure 48. The computing nodes are connected to a dedicated network (ATCN, ATLAS Technical and Control Network) with limited, controlled connectivity to the rest of the CERN network.

To guarantee safe and optimal access for a large number of users to the various software and hardware resources an integrated Access Control System has been deployed (2.11.2) [114]. The chosen approach is based on the Role Based Access Control (RBAC) paradigm, it provides user-based access control in a hierarchical structure and it is used to control the access both at the operating system and application level. The user-roles associations are defined in Lightweight Directory Access Protocol (LDAP) as Network Information Service (NIS) netgroups. A local LDAP cluster based on OpenLDAP software [115] has been setup to be able to keep taking data in case of a forced disconnection from the General Purpose Network (GPN). The user-related information in this standalone system is kept synchronized with the CERN-wide account management, while allowing for additional local accounts (e.g service accounts).

Dedicated application gateways verify and grant the access from remote to specific, well-defined sets of nodes on the basis of the roles and policies implemented. The gateways are implemented on a virtualisation solution, which ensures high availability and manageability for this vital subsystem.

2016 JINST 11 P06008

Two Remote MOnitoring Nodes (RMONs) are connected to the GPN and also have access to RMON-servers (RMON-SRV) connected to the ATCN. The RMONs provide the graphical terminal services for remote monitoring of ATLAS sub-detector systems (2.12.4). All the gateways and the remote monitoring nodes are configured with both host-based and network-based accounting, security monitoring and intrusion prevention systems. A last link between the two networks is provided by the web servers: the GPN web server has limited read-only access to the ATCN web server content. These nodes are exposed to the "outside world" (directly to the GPN, indirectly to the ATCN node), therefore they need special care from a security point of view. Besides the firewall protection, direct access to them is limited to a very few users with special roles.

The TDAQ software is installed (2.17.2) on a NetApp system [144], an integrated solution that enables storage, delivery and management of data. Two Central File Server nodes (CFS) have privileged access to it. The different software areas are redistributed to about 70 Local File Servers (LFSs), which make these available to client computing nodes via NFS. The distribution mechanism is initiated from the CFS. The directories and files are synchronized to the LFSs in parallel and, for increased performance, using a hierarchical grouping of nodes: only a small fraction of the LFSs are synchronized from the CFS and afterwards they serve as "parents" for the other LFSs; the load on the CFS is therefore limited. A bandwidth limitation has been imposed on the file transfer to ensure not to overload the network during this process. Only users with specific roles are allowed to run synchronization jobs on a restricted set of directories by means of the"sudo" command.

Both the CFS and LFSs are standard systems, with the operating system installed on local disks, originally maintained via the Quattor configuration management system [145]. The nodes serving the different control rooms were also standard Quattor-managed systems; as for the file servers, a gradual move towards Puppet [146] has been completed (4.8).

Most of the other ATCN nodes are netbooted clients, i.e. they boot their operating system from the network rather than from a local drive. They are supported by the LFSs, which serve DHCP requests and provide boot images and NFS shares containing the operating system, the TDAQ software, common areas holding log files, different utilities, etc. The configuration of each netbooted client is done via the Boot-With-Me project [147], which allows mounting special areas, running specific scripts at boot time, etc. The rationale behind the choice of the netboot system, and its evolution are described in more detail in 4.8.

Since the winter of 2009, the majority of the nodes connected to the ATCN are running SLC5 (Scientific Linux for CERN [59], a specialized variant of Scientific Linux [148]). At that time the DAQ/HLT software had been built for SLC4, so compatibility between the specific built libraries and the system had to be ensured. Nonetheless, the system has been running smoothly under SLC5.

Security is of course of great concern, so as soon as security updates are available in the CERN IT repository, they are tested on dedicated machines and, if no problems are found, they are applied to nodes connected to the ATCN. Other less vital updates are applied more gradually: they are tested also in the real working environment on non-critical working nodes, in agreement with the sub-detector groups involved. Large scale updates are usually done during scheduled breaks of LHC runs in order not to affect the data-taking process.

A dedicated network that is completely decoupled both from the ATCN and the GPN has also been created for testing purposes ("preseries" cluster, 2.16.2). Its structure reflects the organization of the ATCN, on a smaller scale, and it allows proper and exhaustive testing of a software release before putting it into production.

2016 JINST 11 P06008

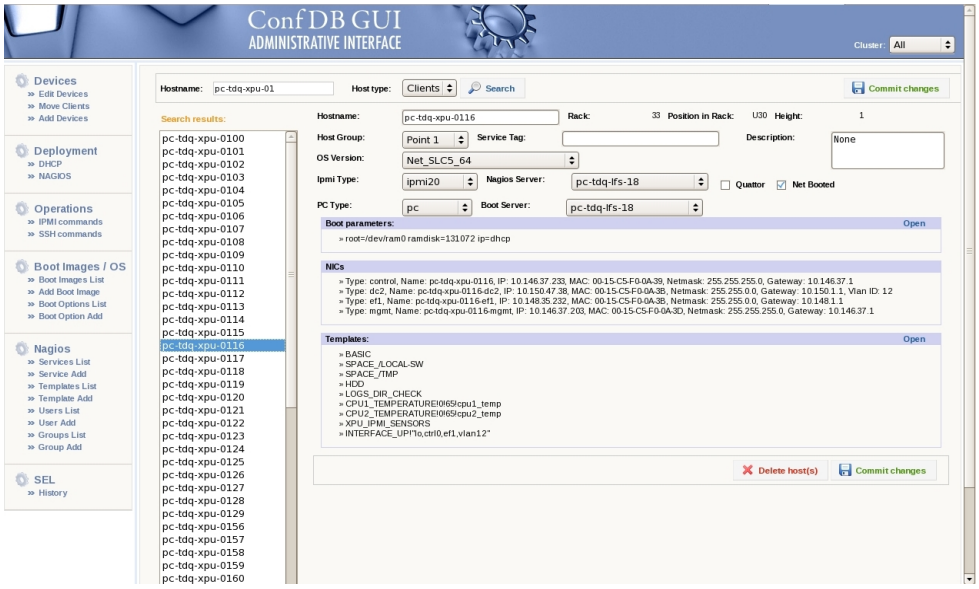


Figure 49. ConfDB user interface.

2.14.2 System administration tools

Node management: the ConfDB GUI and tools. A MySQL [43] database (ConfDB) contains various hardware and software configuration parameters of the nodes. Part of the information (e.g. OS, MAC addresses, location) is retrieved from the CERN central networking database (LanDB), which ensures consistency and simplifies the management. A dedicated set of tools based on this database has been developed and is available to the system administration experts, also via a web-based Graphical User Interface (see figure 49). This GUI allows the handling of a variety tasks; the most notable are the registration of new cluster nodes, the configuration of the Nagios monitoring (2.14.2), the choice of the operating system and boot parameters for the netbooted nodes, and their assignment to a specific LFS, and finally the issuing of IPMI[17] and system commands to custom sets of nodes, with high parallelism.

Monitoring: Nagios. The system is constantly monitored using the Nagios monitoring software [128]. For most of the ATCN nodes only high-level monitoring has been implemented: basic OS warnings and errors are reported, network connections are regularly polled and the hardware state is checked. Specific services, like the NTP[18] service, NFS[19], the DHCP[20] service, etc., are also monitored for the core nodes (LFSs, gateways, CFS, web servers) and an email/SMS alert system has been implemented, so that critical failures are reported to the experts as soon as they occur.

The historical performance data collected by the monitoring system are stored on disks: nearly 30,000 RRD (Round Robin Database) files [112] with a total size of approximately 5.5 GB keep

[17] IPMI (The Intelligent Platform Management Interface [58]) allows the control of the physical state of a node remotely via the network even when the node cannot be accessed via a remote ssh connection.

[18] Network Time Protocol.

[19] Network File System.

[20] Dynamic Host Configuration Protocol: for network interface configuration.

2016 JINST 11 P06008

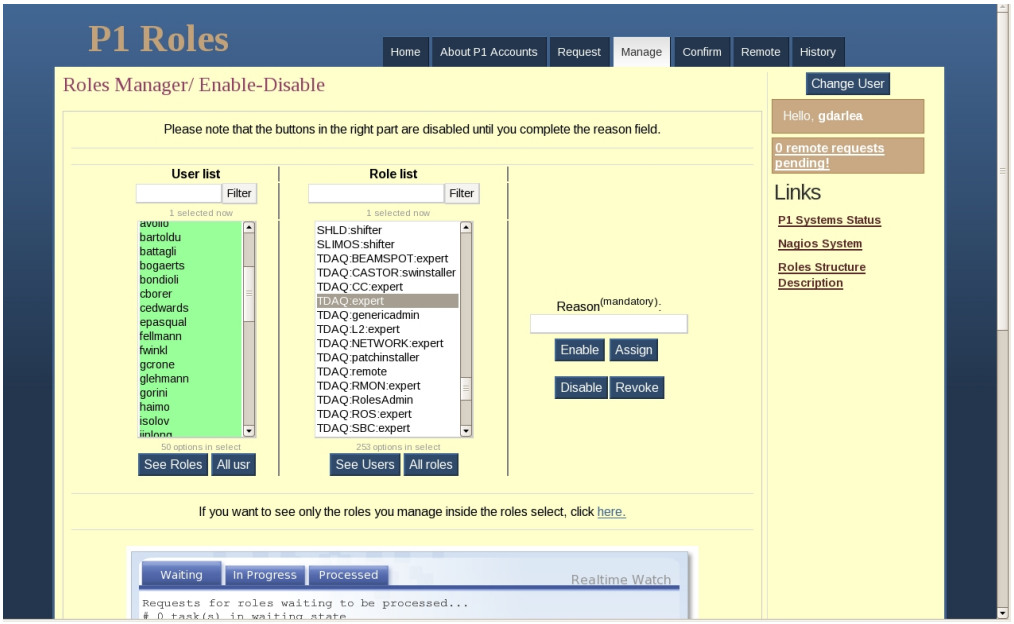


Figure 50. Access Manager Roles Web User Interface.

track of the whole system history. Status information for all the nodes is published automatically in the monitoring section of the ATLAS Operations web server.

There are two access levels for the information available to operators and experts: a detailed view of specific hosts that helps to debug and understand failures, and a general overview of the system. The latter is accessible via the ATCN and the GPN through the corresponding web servers. It facilitates spotting of failures and taking action accordingly and is provided by a set of custom dynamic web pages, which use information stored in a common MySQL database by the separate Nagios instances running on the LFSs.

Role Management: the Access Manager Role GUI. As described earlier (2.11.2 and 2.14) the authentication mechanism is role-based [149]. Therefore a web-based graphical user interface (see figure 50) has been developed to manage the Access Manager roles. The application allows users with specific roles (administrative roles) to manage roles that users need to correctly perform their tasks. All users are allowed to request the needed roles; their requests will be accepted or dismissed by one of the users with administrative roles. The history of all the operations performed is also kept in a MySQL database.

2.14.3 Operational aspects

To protect critical nodes from power cuts of various origins, there are two centralized UPS[21] lines, with diesel generators acting as a backup. About 5% of the equipment deployed in SDX1 is on UPS lines; of these, most are on dual power (so connected both on UPS and on main power). Like other parts of the infrastructure, the power distribution is monitored by DCS [127] and, in case of

[21] Uninterruptible Power Supply.

2016 JINST 11 P06008

failures, automated alarms are sent as SMS message directed to the system administration on-call phone, allowing fast intervention.

In addition to the online statistics, the historical performance data, collected by the monitoring system, offers the possibility of long term post-mortem analysis of the events that led to failures. The logs of core services, such as the Access Manager, LDAP, etc., allow to obtain statistics on the number of connections to the ATCN, web servers hits, etc. A login analyzer tool that offers a detailed and accurate view of the remote access connections inside the ATCN has been developed, as well as a mail log analyzer that offers a complete view of the mail traffic from the ATCN.

2.15 DAQ/HLT operation

2.15.1 ACR and SCR — generic information

The ATLAS data-taking operations are controlled from the main control room, the "ATLAS Control Room" (ACR). The computers in a second control room, the "Satellite Control Room" (SCR), run the same software, usually in monitoring mode, but they can also be used to control the experiment should the main control room become unavailable. The ACR hosts a number of desks from which various operations are performed. The desk relevant to DAQ/HLT is the run control (RC) desk. A second desk, the DAQ/HLT desk was in use until mid 2011. From the RC desk the runs are initiated, the run parameters (such as the data set tags, the set of detectors participating to the current data-taking session etc.) are set, the system configuration database is modified and the messages from the various components are followed with the aim of finding and correcting any issues affecting data taking. The purpose of the DAQ/HLT desk, now replaced by the Shifter Assistant expert system (2.11.9), was to deal with purely DAQ/HLT related issues: keeping track of the health of the HLT farm and of the other computers relevant to the data taking, monitoring the flow of the data to ascertain that various DAQ/HLT components share the load in an equal manner and monitoring the input from the detectors to the DAQ system to spot any "hot" or "broken" detector component.

2.15.2 Operational procedures

The RC desk shifts in the ACR are operated around the clock every day, each shift being eight hours long. The requirements for taking RC desk shifts consist of a day of training and taking at least one shadow shift: an eight hour shift that is performed under the supervision of a more experienced operator. Persons on shift are expected to interact with each other and with experts to operate data taking and to solve problems encountered. For example upon observation of a busy channel from a sub-detector, the relevant sub-detector expert needs to be requested to reset the busy link. An on-call expert is called via the on-call phone carried by the expert on duty if the level of knowledge in the ACR is not sufficient to solve a problem. The main TDAQ on-call experts are members of a team of about 10 people who do not often take normal ACR shifts, but are on duty for longer periods, typically a few days, and solve problems requiring a deeper level of knowledge. If the main TDAQ on-call expert can not solve a problem, a secondary on-call expert with specialized knowledge is called. There are normally 5 secondary TDAQ on-call experts on duty: the ROS hardware expert, the RoIB expert, the data flow expert, the L2 expert and the controls expert. Furthermore operating system and farm hardware problems are tackled by the system administrator on-call expert and data network problems (problems with cables, switches etc) by the network administrator on-call expert.

2016 JINST 11 P06008

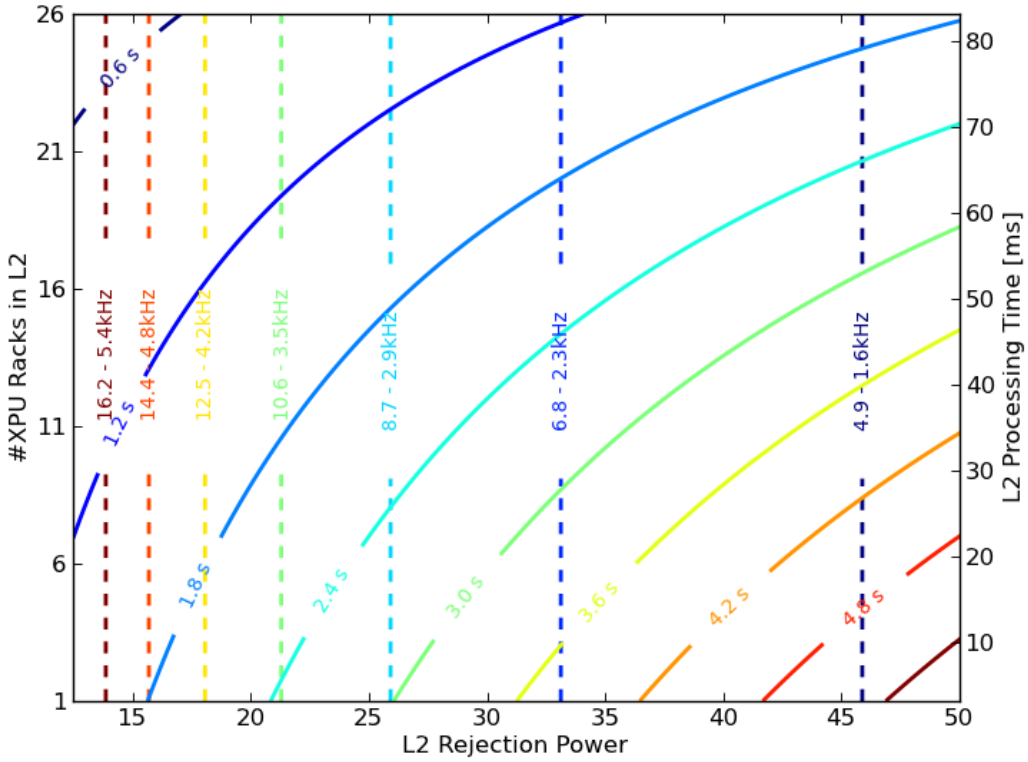


Figure 51. Results of the model of the TDAQ operational capabilities for an L1 rate of 75 kHz, an event size of 1.5 MB, 10 EF specific racks and 27 XPU racks. The curves are labeled with the maximum possible average EF time, the dashed lines are labeled with the EF rejection power and the event building rate.

In 2010 the main on-call TDAQ expert was on average called 4-5 times a week and in a minority of occasions the call was forwarded to the secondary experts.

2.15.3 HLT resource sharing

The ATLAS TDAQ strategy is to deploy computing resources following the ATLAS needs, which primarily depend on the instantaneous luminosity and the trigger menu. In the course of the summer of 2010 ten dedicated EF racks were installed in addition to the existing 27 XPU racks, and the HLT farm reached 50% of its final size. The EF racks were necessary to cope with the announced increase of LHC luminosity. Consequently, the XPU resources were redistributed within the HLT farm, i.e. the number of racks used for L2 processing and therefore also the number of racks used for EF processing was changed. In general the XPU resources were reassigned on a run by run basis, to sustain the evolving rejection factor and processing time of the algorithms.

The maximum number of XPU racks available for L2 processing ($N_{XPU_{L2}}$) can be expressed, if the computing power of the EF racks is not sufficient, as $N_{XPU_{L2}} = N_{XPU_{\text{total}}} - R_{A,L2} f_{L1} t_{EF} / N_{\text{cores}} + N_{EF}$ where: f_{L1} is the L1 accept rate, $R_{A,L2}$ the L2 accept fraction, t_{EF} the maximum value of the average EF processing time per event, N_{cores} the number of cores per XPU rack and N_{EF} the

number of XPU racks needed to provide processing power equal to that of the EF specific racks. The expression allows the evaluation of the data flow working point as a function of: the XPU sharing within the HLT farm, the number of installed specific EF racks, the average L2 and EF processing times and the rejection factors of the algorithms. It has been used to determine the optimal assignment of the XPU resources to L2 and to the EF. Figure 51 shows results obtained for an L1 rate of 75 kHz, an event size of 1.5 MB, 10 EF specific racks and 27 XPU racks, reflecting the HLT resources available at the end of 2010. The number of XPU racks that can be used for L2 processing is plotted as a function of the L2 rejection power ($1/R_{A,L2}$) for a number of choices for the maximum of the average EF processing time (indicated by the labels of the curves)[22]. The maximum possible average L2 processing time can be read from the right y-axis. For fixed L1 accept rate, event building rate and data storage bandwidth the L2 as well as the EF rejection power is also fixed, as indicated by the vertical dashed lines, which are labeled with EF rejection power (inverse of the EF acceptance fraction) and the event building rate associated with a Data Storage bandwidth of 500 MB/s. Starting from the design event building rate of 3.5 kHz and an L1 rate of 75 kHz a rejection power of 21.4 is needed for L2. For a maximum of the average L2 processing time of 35 ms about 10 XPU racks need to be used for L2 processing, so that the maximum possible average EF processing time is 1.8 s. To allow for a longer EF processing time it is necessary to either decrease the event building rate, requiring more L2 rejection power for handling the same L1 accept rate, or to install more EF racks.

2.16 Testing

2.16.1 Testing of new software releases

Overall testing strategy. An important part of the testing strategy consists of testing new software releases or smaller components thereof as much as possible independently of the full detector readout and trigger systems. Large parts of the data flow and trigger software in fact do not directly depend on the ATLAS detector and trigger hardware, or can be tested by means of software emulation of the behavior of the hardware. For example, the ReadoutApplication of the ROS has been equipped with a mode of operation where dummy data can be generated internally. It is also possible to preload simulated data into memory. In this way it is not necessary to connect the computers running the ReadoutApplication to any physical readout link. This strategy allows developers to run tests independently and in parallel. Often the first tests are done on a single host, e.g. the developer's desktop computer. Then testing can move to small clusters consisting of 10–20 hosts, where the behavior of small distributed setups can be studied. Larger tests are done on dedicated testbeds consisting of 50 or more hosts that are equipped with enough resources (CPU, memory, networking, disk etc.) to do some performance testing. Finally release testing is done with the full ATLAS TDAQ system, where it is possible to test the real performance of the data flow system using the actual hardware installed. Often such tests initially use preloaded, simulated data and are followed by essential tests using the actual detector readout, in combination with triggering on cosmic muons or making use of random triggers.

DAQ testing. When a new TDAQ software release is installed, the first test to be executed is related to the control and configuration infrastructure. Real controllers are launched with dummy

[22]The actual average EF processing will be lower.

applications that return success to all state transition requests without performing any action. Once these tests are successful the dummy applications are replaced with data flow applications. In the HLT farm HLT algorithms are not yet run: the actual time (and CPU cycles) spent by physics algorithms is emulated in software. The goal of this second step is to verify the correctness of the data flow from ROS up to the SFOs at the correct speed. Since there are no detectors available at this stage, the ROS sends fragments with randomly generated but correctly formatted event data. The third step involves the inclusion of the physics algorithms and preloading into ROS memories data coming from Monte Carlo simulations or from earlier data-taking sessions. At the second and third stage, the monitoring applications are also tested for their functionality and performance.

Trigger software testing. Changes in trigger configuration or menus are tested in multiple steps. Most testing of algorithm improvements and trigger menu changes is done using the Athena offline analysis software with dedicated signal samples, either simulated or previously recorded, as explained in 2.9.3. For large-scale testing, special datasets of minimum bias collision or simulated data have been filtered to select events passing at least low-p_T L1 trigger requirements. These samples are used for large-scale tests running the trigger in a normal batch queue system using athenaMT.py and athenaPT.py (2.9) to emulate a more online-like environment and check for code stability and rejection power. Typically about one million events are used, corresponding to a few tens of seconds of data taking in the real system. From this processing, changes in L2 data request patterns and trigger processing time can also be found. Final testing is done in a small-cluster partition (2.1.2) test using preloaded filtered data as described above to confirm that results are the same as in the offline environment. This also tests that the full set of state transitions is working correctly.

Tests of new features in the trigger software infrastructure or of major software updates are done using the same procedure as described above, but in addition tests are done using the full DAQ/HLT system with data samples preloaded in the ROS PCs. The full-scale tests catch rare problems, such as certain race-conditions in multi-threaded code, and problems related to many processes starting and running in parallel. While the HLT processing time can be measured for standalone processing, the full-scale tests are the only accurate way to measure the data request latency of L2. This is particularly true for the system running under a significant load. Much testing was done before the start of LHC, running the expected trigger selection at the highest possible rate to find potential bottlenecks. This testing enabled the trigger and DAQ system to run with high efficiency and without major problems during the 2010 collision period, a time period with many changes in the trigger selections and rapidly rising luminosity and therefore trigger rates.

2.16.2 Test platforms

Private small test clusters. Several small to medium size clusters of Linux nodes were built to allow early development and testing of the TDAQ software. These range from clusters of about 10 nodes belonging to a single institute to clusters of 50–80 nodes shared by all interested TDAQ users. Some of these clusters contained hardware that early on allowed tests of ROS hardware as well as network switches together with the DAQ/HLT software. Others were geared towards development and test of the infrastructure of DAQ and HLT, in particular also the trigger algorithms for event selection. These are needed to closely follow the ATLAS offline software, e.g. by providing access to the AFS file system widely used in offline.

2016 JINST 11 P06008

Large scale tests on CERN batch farms. The CERN IT department several times kindly provided access to Linux batch farms of up to about 1100 dual-cpu, single-core nodes (top of the line at the time) for tests of large partitions [150] prior to installation of the DAQ/HLT system. These proved very useful in discovering a few problems that did not show up in other tests, such as algorithms that scaled suboptimally as the number of nodes became large.

The preseries testbed. Before the first set of computing hardware became available a medium sized cluster was built using hardware similar to that planned for the final hardware, the so-called preseries cluster. It supported the use and test of ROS PCs, the RoIB and nodes running the L2SV, L2PU, SFI, EFD/EFPU and SFO applications and included even some parts of the L1 trigger, with a networking connectivity very similar to that of the final system. In 2011 the preseries cluster occupied 6 racks in the first level of the counting house in the SDX1 building and one rack on the lower level of USA 15. The rack in USA 15 contained 12 ROS PCs and their corresponding LFS. The other racks contained 1 CFS, 5 LFSs, 1 LDAP server, 1 MySQL server, switches, 6 nodes for running the SFI application, 2 for running the SFO application, 43 XPU nodes, 30 EF nodes, 4 monitoring nodes (MONs), 2 online nodes, 2 nodes for running the DFM application, 2 for running the L2SV application and one for running the L2RH application.

The file system layout and user access were designed to be similar to those of the full system. Among others, this allowed checks to be made that the installed TDAQ and offline software do not make use of non-local resources not available in the DAQ/HLT system, such as remote databases or the AFS file system (all software needs to be used entirely from local file systems for reasons of performance and isolation). The original preseries cluster proved to be a valuable and well used testbed, and has been upgraded in early 2012.

The DAQ/HLT system as testbed. After installation the final computing hardware also became the ultimate testbed for DAQ/HLT software, although most of the time limited resources like the ROS PCs, CTP, DCS etc. were needed by the ATLAS sub-detector groups. However, before the start of the LHC a sufficient amount of time (often a week at a time) was available to conduct serious functionality and performance tests of the DAQ/HLT system. Gradually, as detector installation proceeded, less time was available, and testing focused on final installation and verification of the DAQ/HLT software. After the start of the LHC, during its running periods, DAQ/HLT testing use is often limited to installation and quick tests of bug fixes. More rarely, when a new major release of DAQ/HLT software needs to be installed, testing time tends to be limited to a day or less. This of course increases the importance of the previously described testbeds.

2.16.3 Testing tools

Setup and testing of large and small partitions (2.1.2) is supported by two tools: PartitionMaker for generating the configuration database (2.11.3), and the runner.py script, which automates the process of running a partition. Once a valid configuration database is available the partition specified by it can also be run interactively using the IGUI. This is often done for testing. However, automatic running of a partition is desirable. The Python script mentioned (runner.py) achieves this by steering a partition through a sequence of state transitions, while keeping track of the results of each step. During the running phase the script periodically collects performance data. At the end, log files from

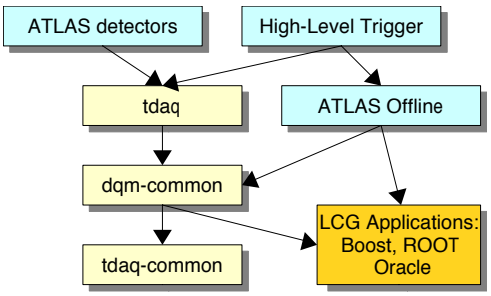


Figure 52. Dependencies of ATLAS software projects. The lighter colored boxes indicate TDAQ projects.

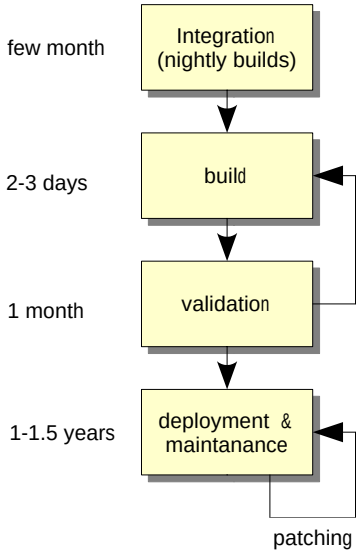


Figure 53. TDAQ software release life cycle and associated timescale.

all applications on all hosts in the partition are collected and written to a summary file. Optionally, the script can also use PartitionMaker to generate a partition at runtime, which it then uses. The runner.py script has been used to regularly run tests verifying the correctness of nightly builds of the TDAQ software. It was also used for commissioning (acceptance testing) of newly purchased racks of HLT compute nodes. This was done by comparing the results to the outcome of known runs using known hardware. Another application is the automation of calibration runs on the basis of electronically generated test signals. For example, the tool has been used to measure threshold curves of the ALFA sub-detector [151], one of the forward detectors, for a number of parameter settings.

2.17 Software installation and maintenance

2.17.1 TDAQ software releases

New versions of the TDAQ software are regularly released. Each release consists of packages built together for a number of platforms (a platform is characterized by a combination of tags, which define the type of binaries built, e.g. *i686-slc5-gcc43-opt* for Intel 32-bit architecture, SLC5 Linux, GNU C compiler version 4.3, optimization enabled). The software is organized in a tree of *projects* or sub-releases: the tdaq-common, dqm-common and main tdaq releases. This factorization allows the integration of TDAQ software with other ATLAS Offline and Trigger software projects as shown in figure 52. There is also a common layer, called LCG (LHC Computing Grid [152]) Applications, which provides a set of third-party packages widely used in ATLAS, like Boost [153], ROOT [133] and CORAL [41]. Version tdaq-03-00-01 of the tdaq project, released in November 2010, consisted of 210 packages, which contain about 1.8 million lines of source code.

The release policy should provide a good balance between further development of the software and stability. Therefore there are two types of releases: major releases and minor releases. Major releases may contain important changes in the architecture of the software, new functionality and API changes in packages, database schema changes and other changes, which may require actions

2016 JINST 11 P06008

from end users (most probably modification of their code). In contrast, minor releases only contain internal changes, which do not require users to modify code. In addition, a patching scheme allows particular problems to be fixed by a binary patch to a package in the release that is already deployed.

In the last years of the development, two to three major releases per year were produced, each typically followed by one to two minor releases. During data taking, no major changes in the software are possible, all maintenance and implementation of new required features is done via the patching mechanism. Typically a new release is installed during a maintenance period. In figure 53 the life cycle of a typical TDAQ release is shown. The integration phase is finished when all the required functionality is available and all packages are successfully built as part of the regular nightly build process. Nightly builds are the main area for the integration of new developments. Automatic validation tests are regularly performed. Once it is decided to deploy a new release it is built and made available for testing using the shared file system and in dedicated labs. The release may be rebuilt including new tags of packages in case a major problem is found in this validation phase. After validation, the release is made available for download and can be deployed. Starting from this point the patching procedures are activated.

2.17.2 Distribution and installation at the experiment site

The release is distributed by means of RPM [154] packages, a de-facto standard for RedHat-derived [155] Linux distributions including Scientific Linux used in CERN. Every TDAQ software package is packaged as an RPM package, such that the RPM version is following the version of the package in the SVN code repository, which allows easy tracking of the versions of the installed software. All RPM packages are available in a number of RPM repositories which were set up to simplify remote installation. At the experiment site the software is distributed as described in (2.14.1), at the end of 2012 2700 packages, containing 4.2 millions files taking 145 GB of disk space, were installed.

2.17.3 Software maintenance and patching

Initially the whole release was distributed as a small number of RPM packages without the possibility to install or update an individual package. This caused problems for the patching policy for TDAQ software, where it is required to have a possibility to easily install a new version of a particular package without disturbing the rest of the software. Furthermore it is also required that roll-back of the patch, i.e. reverting to the previous version of the package, should be possible. With the actual packaging granularity ("RPM per package") a patch for the release is just a new RPM version of a particular package. Thus, a package can be upgraded or downgraded in a relatively short time, which is essential when the intervention to the running system must be done as smoothly and as quickly as possible. It is estimated that the patching dead time, i.e. the time needed to roll-back a "bad" patch, is about 20 minutes.

Given the long lifetime of the production release, the patching procedure is used not only to deploy fixes for urgent issues, but also to introduce some new developments, provided that the public API is not changed and no new behavior is introduced by the patch.

A web-based tracking system based on Savannah [156] is used to manage patching of the software. For every patch, validation on a special testbed is required prior to deploying the patch in the ATLAS TDAQ system.

2016 JINST 11 P06008

(a) Rear view. **(b)** Front view.

Figure 54. ROS racks in USA15.

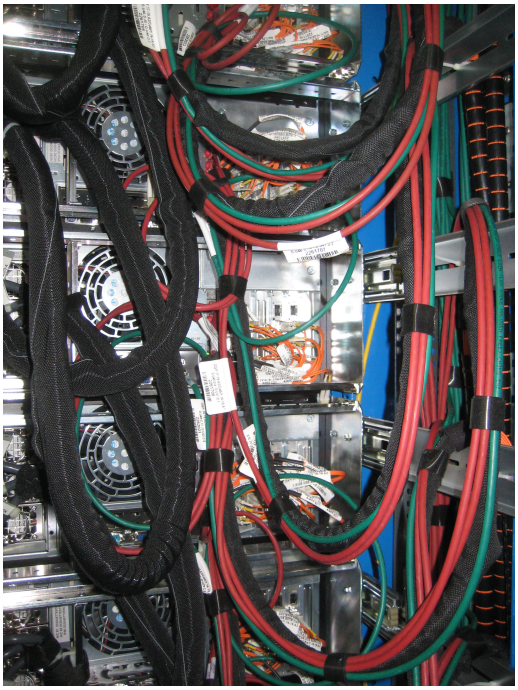


Figure 55. ROS Rack internal cabling: S-link fibers connecting RODs to ROBIN cards in ROS PCs are bundled in shrouds (thick black "cables"), individual fibers (orange) connecting to ROBINs can be seen in the centre of the photo. The red and green cables are copper GbE cables connecting to a Control Network and to a DataCollection Network "pizza box" concentrator switch respectively.

Since October 2009 until the end of Run 1 (February 2013), more then 401 patches were installed in the production system without changing the base TDAQ software release.

2.18 Hardware infrastructure

All TDAQ hardware is mounted in racks, placed either in the underground area USA15 or in a barrack, serving as counting house, in the SDX1 building (see figure 2 for the locations of USA15 and SDX1). The ROS PCs, including those of the preseries testbed (2.16.2) and a few hot spare ROS PCs, the "pizza box" switches associated with the ROS PCs, and the RoIB are contained in racks in USA15. The 16 racks with the 151 ROS PCs used in the DAQ system are positioned close to the racks with the RODs of the sub-detectors they serve. All other hardware of the TDAQ system is installed in racks on the two levels of the barrack in the SDX1 building.

2.18.1 USA15 racks

Racks in USA15 have a 52U usable height and are powered from overhead Canalis [157] system busbars and Twido [158] controllers which can be seen above the rack in the rear view of figure 54. The racks were designed for vertically cooled chassis based systems which is not appropriate for the

2016 JINST 11 P06008

current industry standard of horizontal cooling for PCs and switches. All the TDAQ racks are thus fitted with rear door mounted, water cooled, heat exchangers with a capacity of 9.5 kW per door. In practice the ROS machines are constrained more by volume than power since a maximum of 12 ROS PCs can be mounted per rack and each PC has a power consumption of between 300 and 600 W.

Power distribution within the rack is done with socket strips fed from the Canalis busbars. Each socket strip has four groups of three sockets and each group is sequenced during power up. Mechanical relays stagger the power up of each group by 100 ms in an attempt to avoid the inrush current tripping the supply. Unfortunately the inrush per group was still enough to damage some of the relay contacts which fused into an 'always on' position. In the event of further problems the racks will be equipped with thermistors as described in 2.18.3.

Ethernet switches are mounted at the top of the racks and supported by custom made rails to allow for maximum air circulation. The current generation of switches do not have front to back, or even back to front, circulation and thus are not readily compatible for operation within an enclosed rack. Special attention to the airflow around the switches was necessary to ensure adequate cooling.

Cable management is a serious issue since the PCs are required to be extracted for maintenance while still connected and functional. There are typically 12 S-link fibers, 3 Ethernet connections and 3 power cables per ROS PC and the PCs, which are mounted on telescopic rails, need to slide out 0.7 metre while still operating. Close attention was paid to avoid blocking the air flow with the resulting cable harnesses (see figure 55). Running experience shows that the exhaust air temperatures before the heat exchanger never exceed 30 °C for an ambient temperature which rarely varies from a nominal 21 °C.

To keep the ROS PCs running if there is a problem with one of the 230 V phases (R, S, T) provided by the Canalis busbars the triple redundant (2 out of 3) power supplies of the ROS PCs have been connected in such a way to the socket strips that each PC receives power from all 3 phases.

The ROS PCs are equipped with air filters at the front. Inspections in 2011 and 2012 showed that even though the filters have never been replaced since the PCs were installed in 2006 the inside of the PCs were still free of dust. The filters of some PCs that were installed very early are visibly loaded with dust but are still sufficiently transparent to air and no significant increase of the temperature level inside the PCs has been observed. The ventilation of the PCs is performed by 3 chassis fans (two pushing and one pulling), one CPU fan and 3 fans in the PSU modules. All temperatures (CPU, motherboard, PCI cards) are at a very low level, even with the hardware in full operation.

2.18.2 The SDX counting house in the SDX1 building

The counting house, in the form of a two floor barrack, in the SDX1 building [159], housing HLT/DAQ equipment, has been designed and built by the CERN Technical Support division. The size of the barrack is constrained by a crane, the shaft to USA15 and existing walls of the SDX building. It can accommodate up to 100 racks of dimension 600 mm x 1000 mm and is known conventionally as "SDX".

The load bearing structure of SDX is shown in figure 56. There are three pairs of rails running lengthwise on each floor (shown in red) on which the racks are mounted directly. The cross beams (light blue) reinforce the structure giving a floor loading capability of up to 800 kg per rack load. The racks are industry standard Server Racks (e.g. RITTAL TS8). To avoid overloading level 2 the racks are limited to 47U. Although the maximum loading is the same for both levels there is the

2016 JINST 11 P06008

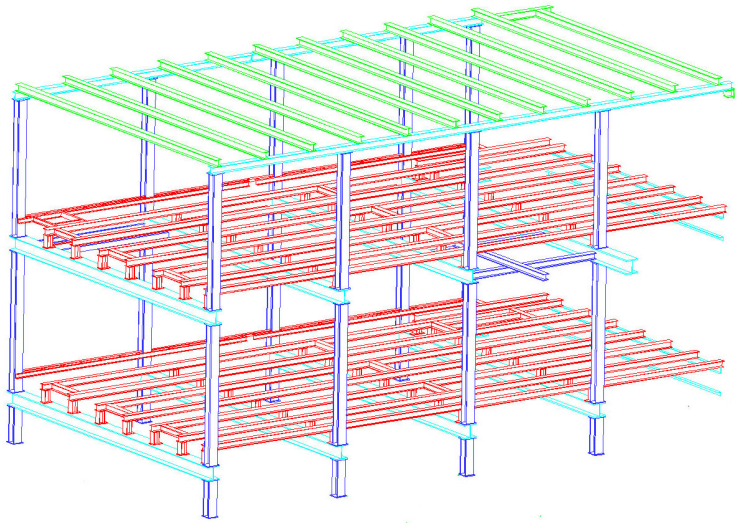
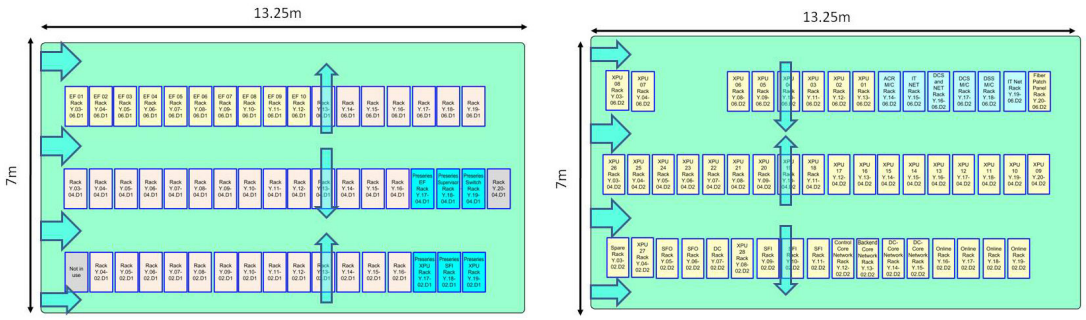


Figure 56. Load bearing structure of SDX.



(a) SDX Racks: Level 1. **(b)** SDX Racks: Level 2.

Figure 57. SDX floorplans.The arrows indicate the direction of the air flow.

possibility that the lower level could be given additional reinforcing and to take advantage of that option the racks on the lower level are 52U high. However, they cannot be fully loaded unless and until that reinforcing has been provided. Each rack has a water cooled rear door mounted unit with a cooling capacity of 9.5 kW for the racks on the second level. The racks on the first level have a capacity of 15 kW in view of the optional higher loading.

The racks are distributed according to the plans shown in figure 57. The dimensions of each level highlight the limited space available to house upwards of 2000 computers. Level 2 is fully occupied with 27 XPU racks (see figure 58) and 16 racks with other TDAQ related equipment (SFIs, SFOs, online and monitoring machines, infrastructure servers, switches). In addition there are 3 racks of Detector Control System (DCS) and Detector Safety System (DSS) machines that serve the whole experiment plus a rack of Control Room servers. The lower level is partially occupied with 10 racks of EF processors and the 6 racks of the preseries testbed. The remaining racks await the future deployment of more EF processors. In 2011 and 2012 4-blade servers, each inside a 2U enclosure and with a power consumption of 1100 W nominal, have been installed, 8 or 10 per rack.

2016 JINST 11 P06008

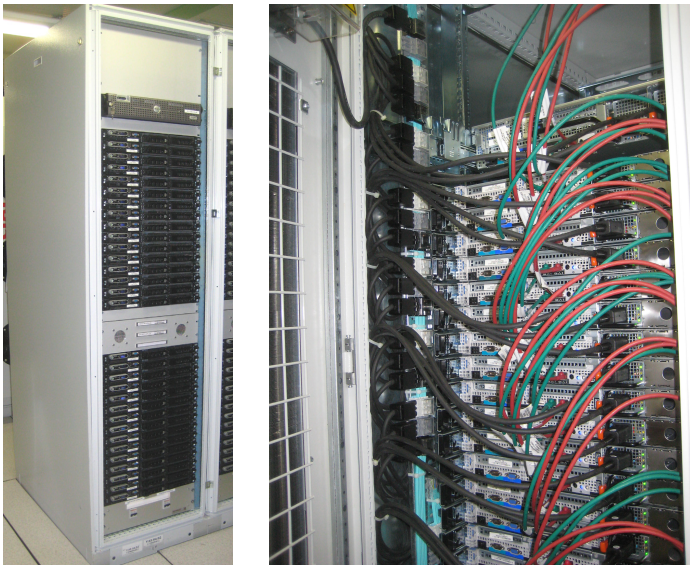


Figure 58. Left: front view of XPU rack (SDX Level 2) with 31 1U XPU machines, of which one is used only for monitoring and control, one local file server (at the top) and two "pizza box" concentrator switches (behind the panel in the middle), one connects to the Control Network, the other to the DataCollection as well as the BackEnd network (2.10.1). Right: cabling in the back of an XPU rack. The green and red cables are GbE cables connecting the nodes to the concentrator switches.

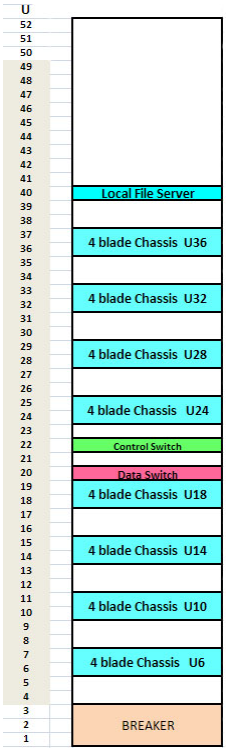


Figure 59. Installation of 2U 4 blade servers in an SDX Level 1 rack.

Therefore the total consumed power is similar to the previous generation on a per rack basis, but it is concentrated into a quarter of the volume. To spread the load more effectively over the height of the heat exchanger the processors are mounted according to the layout shown in figure 59. The shaded area to the left indicates where the heat exchanger grid fits over the rear door.

Each level of SDX is 2.67 m high giving a per level volume of 248 m^3. Of that about 80 m^3 are occupied by the racks themselves. Because there are an odd number of rows, with such a narrow space between them, it was not possible to simply use a classic 'hot and cold aisle' air cooling arrangement. Typically air cooling requires a large volume of air around the rack to avoid hot spot development and in practice will only cope with power levels of 4 to 6 kW per rack. The processor farm racks typically produce 6 to 9 kW. So the racks were designed to be water cooled with rear door mounted heat exchangers that remove up to 90% of the generated heat. The remaining 10% is handled by air cooling. The resulting airflow pattern is indicated in figure 57 with blue arrows. The racks are mounted with airflow in different directions between rows and between levels because of differences in the way that the water supply pipes are laid. Both levels however have cold air arriving along one end wall, to the left in this figure, flowing down the length of the room and leaving from the opposite end wall. The air conduits external to the room can be seen in figure 60 where the incoming cold air conduits, connecting to the rear part of the barrack, are in blue and the exiting warm air conduits, connecting to the front, are in pink. The combination of air and water cooling has been shown to be adequate for periods with high external temperatures and all machines

2016 JINST 11 P06008

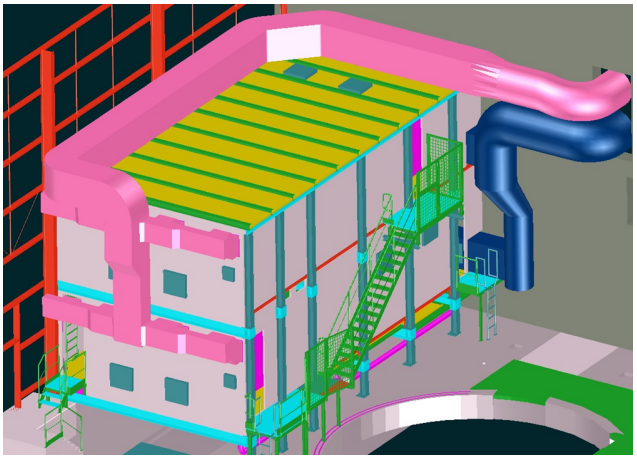


Figure 60. The SDX counting house in the SDX1 building with the conduits for incoming cold air (in blue) and outcoming warm air (in pink).

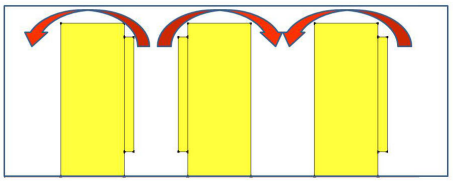


Figure 61. Turbulent air flow rising above the rear door heat exchangers and passing over the top of the racks.

powered up and running with maximum CPU utilization. Water ingress/egress temperatures are typically 13/16 °C and the air ingress/egress temperatures are typically 17/24 °C.

The asymmetric flow through the racks produces a secondary effect that was only noticed after the installation. Onto the air flow down the length of the aisles is superimposed a turbulent flow that rises above the egress of the rear door heat exchangers and passes over the top of the racks into the ingress aisles. This flow is shown schematically in figure 61. The aisle between the central and right hand rows collects more than the left hand aisle and also this is where more of the maximum load racks are to be found. As a result the maximum ambient temperature at the ceiling of this aisle can be 3 to 4 °C higher than the maximum in the left hand aisle. This had to be taken into account when establishing safe working limits.

2.18.3 Power distribution in SDX

The development of the power distribution in SDX is described in ref. [159] and was based on an estimated load of 11 kVA per rack delivered on three phases of 16 A each. With an estimated power factor correction of 0.9 this results in an upper bound of 10 kW converted to heat. This matches approximately the chosen water cooled capacity of 9.5 kW for an inlet water temperature of 15 °C on the second level, while it is below the cooling capacity of 15 kW on the first level.

The circuit for power distribution on each floor of SDX is shown in figure 62. There are three distribution panels per floor and each panel controls one row of racks. Inside each panel there are four rows of ten, 3-phase 16 A breakers. Each row is controlled by a 3-phase 100 A breaker. Each rack is controlled by two of the 16 A breakers so each row of breakers controls up to five racks. There is thus provision for up to 20 racks per row of racks. This arrangement defines the limits on power consumption, and the relative distribution of power among groups of racks.

Any one rack could theoretically consume up to 2 x 3 x 16 x 230 = 22 kW but this is not likely to be reached since the limit on any group of five racks is 3 x 100 x 230 = 69 kW. So per row there is a limit of 280 kW and per floor the limit is 840 kW.

2016 JINST 11 P06008

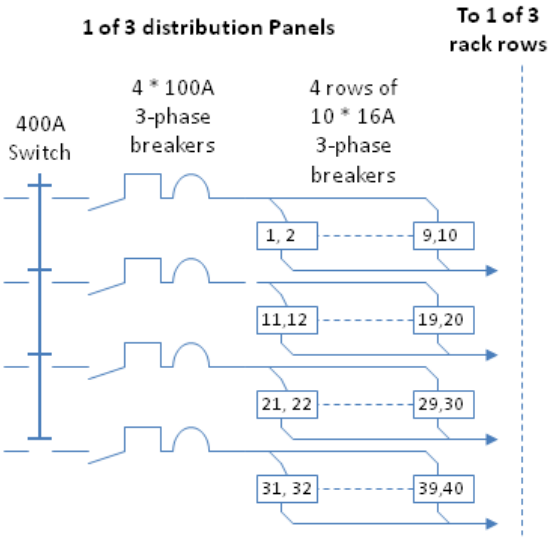


Figure 62. SDX Power Distribution.

Figure 63. Ecobus® bus and connector technology.

It was recognized in the original design that the expected inrush current would trip any standard 30 mA differential breakers installed on a per phase, rack basis. Power distribution within a rack is thus done without such protection and therefore cannot use any sockets into which standard consumer equipment might accidentally be connected. The solution was to use the Ecobus® Technology for all the internal power distribution as shown in figure 63.

The power supply to each rack is controlled with a manual switch in the rack itself and these are fed from protected breakers mounted in the distribution cabinet. These breakers incorporate two distinct over-current protection mechanisms: magnetic and thermal. Rapidly rising surge currents that significantly pass the current rating of the breaker will be reacted to magnetically with a load/time response defined by the "curve" rating of the breaker. We chose the most robust "D" type curve devices which will trip for 10 to 20 times nominal current. In practice however the first generation XPUs far exceeded this inrush value and systematically tripped the breakers when all were powered on together. The solution was to add high current rated thermistors [160] in series with the manual breakers in each rack repartition boxes. The assembly is shown in figure 64, the circuit in figure 65. The thermistors are the black components mounted either side of the manual switches.

These devices have high resistance at nominal room temperatures (about 2.5 Ω), falling as the device temperature increases to about 0.03 Ω at the maximum rated current of 15 A. Cold-start currents were thus limited to < 90 A per phase for the few milliseconds of overload. The D-curve response time for a factor 6 of overload is more than 1 second and so the trip is avoided. As the thermistors heat up, their resistance falls until they reach a stable and auto-regulating operating point where they dissipate just enough heat to keep them at a low resistance point.

The thermal part of the trip functions if there is sustained operation beyond the nominal operating point. This is a slow process for small excursions, typically, for up to less than 10% overload for example, this can be from minutes up to several hours.

If the breakers do trip for any reason they need to be reset manually which is only done after finding and repairing the root cause of the failure. As long as the breakers are set it is always

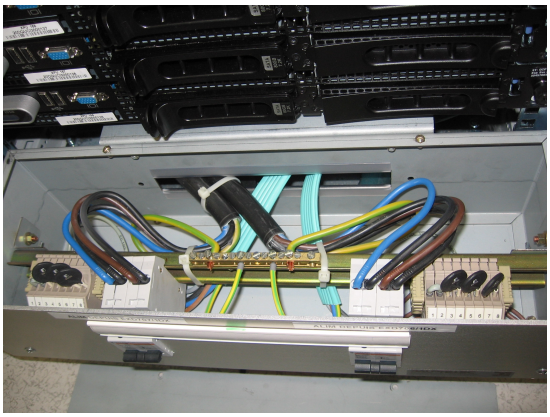


Figure 64. View of the interior of a repartition box.

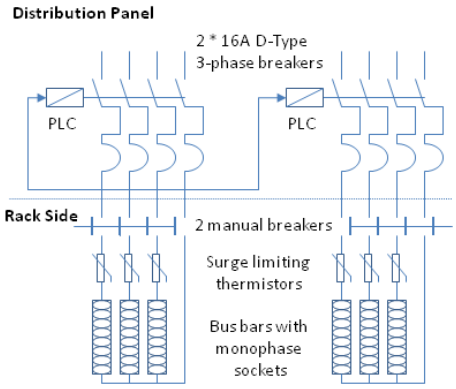


Figure 65. Rack supply circuit.

possible to open or close the circuit to the rack by remote commands which are interpreted by the Programmable Logic Circuit (PLC) shown in figure 65.

Having a common neutral to the three phases inside a rack exposes the risk of an 'open neutral'. If the neutral return is broken for any reason then the common star point will assume a potential that is a function of the vectorial sum of the three phases and their respective loads. Only if the loads are balanced will this be zero. During commissioning one breaker suffered damage and upon being reset the neutral remained open. The rack in question was only partially filled and the loads unbalanced. Six servers and two switches suffered damaged power supplies as a result. No other instances of this have been observed.

2.18.4 UPS

There are four demands for uninterruptible power (UPS) in SDX:

- Online and monitoring machines on level 2 of SDX are controlling processes in USA15.
- Events in transit need to reach permanent storage in the nodes running the SFO application and their file systems need to be closed cleanly before the host machines may be shut down.
- SDX houses safety and security machines belonging to DCS and the main control room display machines.
- The core IT routers maintain essential connectivity between the experiment site and the rest of the CERN networking infrastructure.

UPS is provided and is currently operating at the budgeted limit of 36 A per phase.

2.18.5 Safety and protection

The two levels in USA15 and the two levels in SDX occupied by TDAQ racks are each monitored by DCS [161] for smoke, air temperature, inlet cooling water temperature and ambient humidity. For each of the variables DCS will report on three increasing levels of alarms: warning, error and fatal.

The small air volume and high power dissipation in SDX makes it very vulnerable to cooling failures. Occasional cooling interruptions have shown that if cooling fails and power remains on,

the ambient temperature will rise at 1 °C/min. To prevent shutting down prematurely in the event of a partial cooling failure and to avoid the overhead of a full scale power down, an automated sequence of sequential load shedding has been established to shut down first the HLT processors, then the event output processors and finally leaving just machines powered on UPS to continue. The aim is to stop data taking but to allow for a clean shut down of all processors and thus safely save all accepted events. If shutting down racks in this fashion does not arrest the continued rise in temperature then DCS itself will cut power to all non-UPS racks. If all these efforts fail, then the ATLAS Detector Safety System (DSS) will trigger at 55 °C and cut all power to SDX.

The racks supplied by the UPS are a non-negligible load of the order of 25 kW and will need some cooling to keep temperatures at an acceptable and sustainable level. If there is a cooling failure then those racks can also be fed from an alternative, non-recycled water supply which, while not chilled is still sufficient to cool them.

Since the point of humidity monitoring is just to avoid condensation on the cooling system, the warnings are based on the difference between the ambient humidity and the inlet water temperature rather than any absolute values.

The individual racks themselves are monitored for water outlet temperature, air temperature inside the rack, fan and breaker status. Water leakage detection is also being installed to cover common zones where TDAQ racks are placed. Alarms at this level are distributed over SMS to responsible persons and with warnings to the Shift Leader In Matters Of Safety (SLIMOS). Interventions are manual; no automatic procedures have been envisaged.

3 Results of performance tests and observations from data taking

3.1 ROS performance tests

To determine the maximum rates that can be handled by the ROBINs and by the ROS PCs extensive tests in dedicated test setups were performed. In addition the tests were used to find the conditions for obtaining maximum rates. Four rates have to be distinguished: (i) the L1 accept rate, this is the rate with which event fragments arrive via the ROLs and deletes are received via the network,[23] (ii) the rate of L2 requests, (iii) the rate of L2 E_T^{miss} requests[24] and (iv) the rate of Event Builder (EB) requests. L2 or EB requests are sent to the ROS PCs via the DataCollection network and should result in building an event fragment from fragments stored in a subset of the ROBs or all ROBs, respectively, followed by forwarding it to the DataCollection network. The attainable maxima of these rates depend on each other, the sizes of the event fragments, the grouping of delete requests, and the configurations of the ROBINs, of the ROS PC and of the ReadoutApplication (the application running on the ROS PC).

3.1.1 Performance of the ROBIN

The performance of the ROBINs was studied with a small test program that determines the maximum request rate for a given fragment size and delete rate. Data received are immediately discarded.

[23] Deletes are sent in groups, so the message rate is smaller than the L1 accept rate by a factor equal to the number of deletes per group, which is typically 100.

[24] This type of requests, used for the second-level missing energy trigger for requesting energy sums from the calorimeter ROS PCs, was introduced in 2012 (2.3.4).

2016 JINST 11 P06008

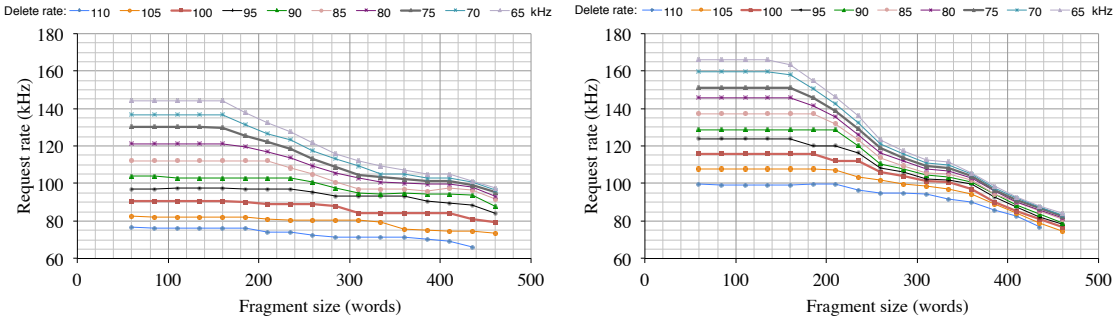


Figure 66. *Left*: standard ROBIN with PCI interface, *right*: ROBIN with PCIe interface: maximum request rate for different delete rates and as a function of the fragment size (a word contains 4 bytes). Input data were supplied via the S-link inputs using a DOLAR card (see text). There are no measurement results for a delete rate of 110 kHz and a fragment size of 460 words because the S-link bandwidth of 200 MB/s is exceeded.

An upgraded ROS PC (3.1.2), equipped with ROBINs, was used to run it. The effect of the test program on the results of the measurements is negligible. Input of data in the ROBIN has been done using the internal data generator of the ROBIN or with the help of a special test generator, the DOLAR.[25] The latter is a PCI card with 4 S-link source interfaces and capable of generating correctly formatted event fragments, all with the same size but with monotonically incrementing L1Ids. The test program generated requests and deletes as fast as possible, respecting a maximum number of outstanding requests. The deletes were forwarded in groups of 100 to the ROBIN, separately for each ROB. The request and associated delete rate was throttled if the responses of the ROBIN indicated that data requested were not yet available. There is no need to distinguish between L2 and EB requests, as separate requests for each ROB are sent to the ROBIN, only the total request rate matters. The ratio of deletes and requests were varied until a target delete rate was observed.

Figure 66 shows characteristic results of the ROBIN performance tests, obtained with the version of the ROBIN installed in the TDAQ system (left plot) and with a new version of the ROBIN with a PCIe interface, described below. A DOLAR has been used as external data generator for these measurements. In this case the S-link flow control throttles the rate with which the DOLAR sends event fragments once the buffers of the ROBs have been filled. For small event fragments there are two quantities that determine the behavior: the time needed by the CPU of the ROBIN to handle an incoming fragment and the CPU time needed to handle a request. From a linear fit to the delete rate plotted as a function of the request rate for a fixed small fragment size the CPU time needed for handling a single incoming fragment (including the CPU time needed to free the buffer page it is occupying) can be obtained. For the standard ROBIN it is found to be 2.1 μs. Hence for data arriving via all 3 ROLs 6.3 μs is needed, resulting in an upper limit for the L1 accept rate of 160 kHz. For handling a request 4.1 μs is needed. At 100 kHz L1 accept rate this results in an upper limit of the request rate of 90 kHz, as also can be seen in the left plot of figure 66. For larger fragments it seems that because of the longer time needed for transfers across the PCI bus, the transfers and processing by the CPU no longer proceed completely in parallel and the maximum request rate falls. The maximum available bandwidth for transfer of data from the

[25] By replacing the firmware a FILAR S-link destination card [65] can be transformed into a DOLAR card.

ROBIN memories, via the PCI bus, to the memory of the ROS PC, is 266 MB/s. In practice for the largest fragment sizes of interest a maximum throughput of about 180 MB/s is observed. The performance of the ROBIN depends on the relative rate of servicing the Used Page FIFOs, the Free Page FIFOs and the Message Descriptor FIFO and associated DPM (2.3.3). The results shown in figure 66 have been obtained after tuning the relative rates to provide the maximum request rate. The ROBIN firmware also allows deletes for all three ROLs to be combined when sent. This is not supported by the ReadoutApplication but would result in an improvement of the maximum rate that can be handled by about 10% for 100 kHz delete rate.

The dependence of the maximum request rate on the number of deletes per group was also studied. For a group size of 50 and a delete rate of 100 kHz the maximum request rate is at maximum 10% lower than for a group size of 100. For a group size of 10 and for the same conditions the reduction in maximum request rate is about 50% compared to a group size of 100. Group sizes larger than 100 are not supported by the firmware of the ROBIN, the results show that these would not result in substantial higher maximum request rates.

In view of the expected obsolescence of the PCI bus a variant of the ROBIN with a PCIe interface has been designed and a prototype series of 10 boards has been produced. The PCIe interface is software compatible with the PCI-X interface and is provided by a PLX PEX 8311 [162] bridge, which implements a 1-lane Gen1 PCIe interface. A similar, software compatible, 4-lane bridge was not available at the time of design of the board. Although the theoretical throughput of the 1-lane interface is 250 MB/s in practice a somewhat lower throughput, 150 MB/s, than the maximum throughput of 180 MB/s of the original version of the ROBIN has been observed. This causes the maximum request rates for the largest fragment sizes (350 or more words) in combination with delete rates lower than about 100 kHz to be lower than for the PCI ROBIN, see the right plot of figure 66 for measurement results. However, the processor of the PCIe ROBIN has a clock frequency of 667 MHz, while this is 400 MHz for the PCI ROBIN. For smaller fragments therefore higher request rates are observed, for example for 100 kHz delete rate the maximum request rate increases from 90 kHz to 115 kHz, as can be seen in figure 66. For small fragments the CPU time needed for handling an incoming fragment is 1.9 μs, giving an upper limit of the L1 accept rate of 175 kHz, for handling a request 3.8 μs is needed.

3.1.2 Performance of the ROS PC

The performance of the ROS PC is determined on the one hand by the ROBINs and on the other by the combination of the hardware of the PC and the software running on it. The performance of the standard ROS PCs, installed in the TDAQ system at the start of Run 1, is mainly determined by the latter as will be obvious from the measurement results to be discussed. The configuration of the hardware and of the operating system affect the performance of the ROS PC, as well as the configuration of the ReadoutApplication. For the first category the version and the type of Linux kernel used (uniprocessor or SMP[26]), the use of hyper-threading, the use of interrupt coalescence for the Ethernet interfaces, and the use of safety features of Linux (SELinux) are important. It has been found that the best performance for the standard ROS PC, equipped with one single core CPU (Irwindale [53]) obtained with a uniprocessor Linux kernel as was available in SLC4 [59])

[26]Symmetric MultiProcessor, for use with a CPU or CPUs with multiple cores and /or supporting hyper-threading.

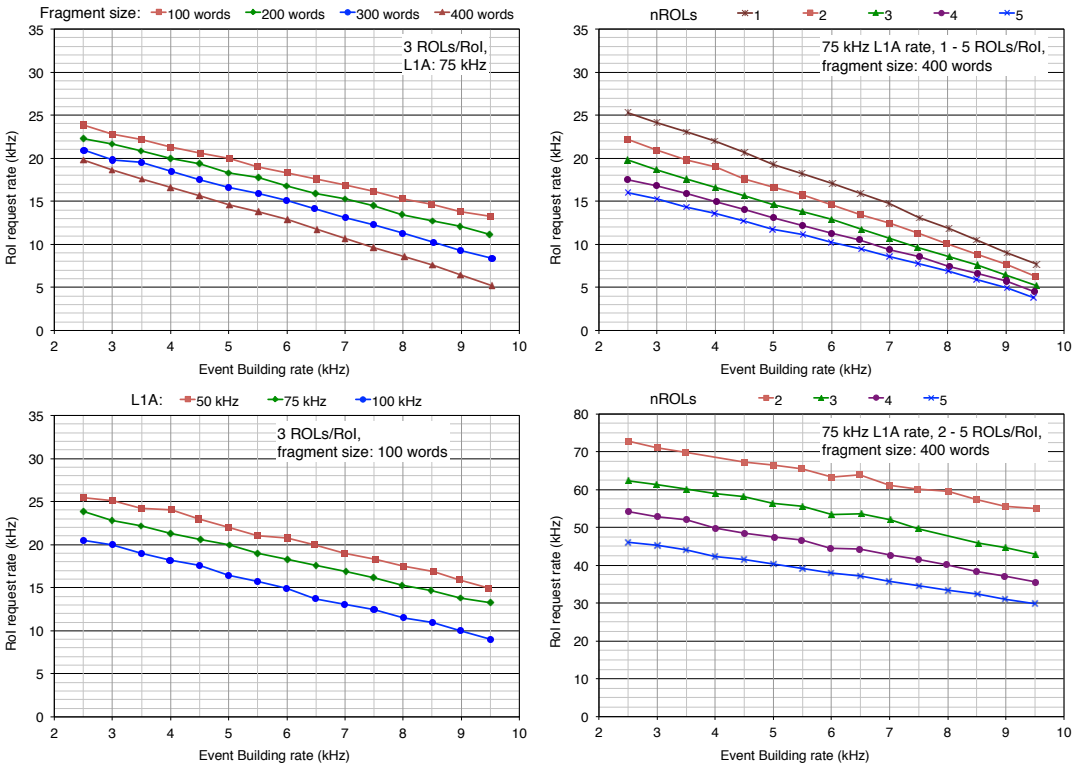


Figure 67. ROS PC performance: maximum RoI request rate as a function of the event building rate (no L2 E_T^{miss} requests). *Top left*: standard ROS PC, optimized, data is requested from 3 ROBs per RoI, results for four fragment sizes (a word contains 4 bytes). *Bottom left*: same as top left, but for a fragment size of 100 words and for different L1 accept rates. *Top right*: same as for top left, but for a fragment size of 400 words and for different number of ROBs per RoI request. *Bottom right*: ROS PC with new motherboard and CPU and using all 4 ports of a Silicom PEG-4 GbE interface card (installed instead of a Silicom PEG-2i) for a fragment size of 400 words.

and hyper-threading switched off in the BIOS of the PC. The reason for this is probably that no real parallel processing is possible, so that the kernel uses thread synchronization primitives that have less overhead than otherwise would be possible. However, the SLC5 Linux distribution [59] does not include a compiled uniprocessor kernel, therefore using the SMP kernel is preferred. In this case the best performance is achieved by switching hyper-threading on in the BIOS and running the ReadoutApplication on both hyper-threads. The performance can be further optimized by tuning the interrupt coalescence parameters of the network interface card and disabling SELinux. The latter is possible as the environment in which the TDAQ system is operated is sufficiently protected.

In figure 67 results of test measurements are presented. These measurements were done with a ROS PC with a standard configuration (with 4 ROBINs and a 4-port GbE interface card installed) and typically two PCs each running a special test program and each directly connected via a dedicated GbE link to the ROS PC. Each test program generates "RoI request" messages, "EB request messages" and delete requests and sends these via the Ethernet link to the ROS PC, the ROS PC outputs requested data via the same link. The ratio of the number of request messages with respect to the number of delete messages is set at the start of a test run. Each test program generates requests

2016 JINST 11 P06008

and deletes only for L1Ids equal to i + n, with i equal to a unique number given to the test program and in the range from 0 to the number of test programs, and n incrementing from 0 with a step equal to the number of test programs. The RoI request messages retrieve data from a fixed number of ROBs, the number is also set at the start of a test run. The ROBs from which the data is requested are chosen at random for each request. EB request messages retrieve data from all ROBs in the ROS PC. The data itself is generated by the internal data generators of the ROBINs, the fragment size is also set at the start of a test run. These data generators transfer fragments to the buffer memories on the ROBIN cards at maximum speed of 266 MB/s and are throttled once there is no memory space left. One of the test programs acts as master and communicates the ratios of RoI request messages and of EB request messages with respect to delete messages via the control network, to which all PCs are connected, to the other test programs. The delete requests are sent in groups, the group size can be chosen, the standard value is 100. The test programs send requests as fast as possible, but limit the number of outstanding requests. The result of a test run is the delete rate observed. It is possible to set a target delete rate and to initiate test runs automatically, where at the start of each run either the ratio of RoI requests and delete requests or of EB requests and delete requests is adjusted until the observed delete rate is within a certain margin equal to the target delete rate. In this way the request rates compatible with the delete rate chosen can be determined. The setup consisting of ROS PC and PCs each running a test program is controlled from another PC, connected to the control network mentioned. All PCs are running the TDAQ software, the standard configuration and control facilities are used.

The plots in figure 67 all show the RoI request rate (L2 request rate) as a function of the EB request rate. The left top plot shows the effect of changing the fragment size, the left bottom the dependency on the delete rate. The average delete rate is equal to the L1 accept rate during data taking. Both plots are for 3 "ROLs per RoI", i.e. the number of ROBs from which data is requested per RoI request. The results show that for a nominal EB rate of 3.5 kHz, an L1 accept rate of 75 kHz and 1–5 ROLs per RoI and 400 word fragments, the maximum RoI request rate has a value between 14 and 23 kHz, for smaller fragments these values are higher. For 100 word fragments the maximum RoI request rate is of the order of 25% higher. Increasing the L1 accept rate from 75 kHz to 100 kHz results for 100 word fragments in a reduction of the maximum RoI request rate by about the same fraction. The rates observed are only obtained if the configuration of the ROS PC is optimized as discussed (ReadoutApplication running on both hyper-threads, interrupt coalescence optimized, SELinux disabled, no CRC checksum checking of a fraction of the data requested).

For some ROS PCs the maximum rates were by 2010 close to rates predicted for data taking at the maximum luminosity expected in 2011. Because of this and the ageing hardware of the original PCs it was decided to exchange the motherboard, CPU and memory of each ROS PC with a new type of motherboard, allowing the use of a faster multi-core CPU and faster memory. Results of the same type of performance measurements for the new hardware are shown in the bottom right plot. The ReadoutApplication was running on 2 of the 4 cores of the 3.00 GHz quad-core CPU (using more than 2 cores does not result in higher rates) and 4 GbE links were used in conjunction with 4 test programs. The configuration of the ROS PC was optimized as described before. For four ROLs per RoI and fragments of 400 words the output bandwidth required is about 400 MB/s, so the existing network infrastructure (with 2 GbE links per ROS PC) will determine the maximum rate in this case. For example for 3.5 kHz EB rate this rate is 27 kHz instead of 52 kHz if 4 GbE links can be used, still appreciably higher than the 16 kHz observed for the standard ROS PC. Apart from the

2016 JINST 11 P06008

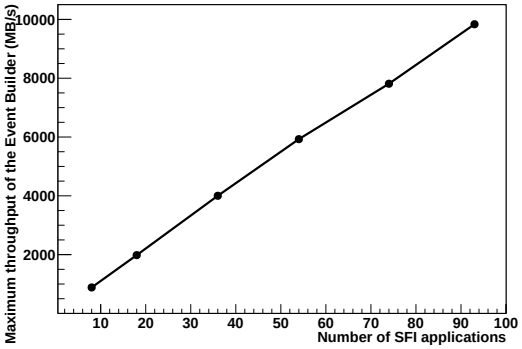


Figure 68. Maximum throughput of the Event Builder as a function of the number of SFI applications.

required extension of the network infrastructure, using 4 GbE links per ROS PC would also require modification of the TDAQ software. Tests done with the preseries testbed have shown that this is not necessary if link bonding [81] is used. Unfortunately it was found at the end of 2011 that loss of network connectivity occurred, albeit with low probability (of the order of once per week for about 100 PCs), for the Silicom PEG-4 4-port interface card [56] if used in combination with the X7SBE motherboard. This problem did not occur after replacement of the PEG-4 cards by PEG-2i [57] cards in the PCs of which the motherboards have been replaced. However, since the PEG-2i cards only have 2 ports these ROS PCs could no longer connect via 4 GbE ports to the data flow network, but future replacement by 10GbE interfaces was foreseen.

3.2 Event Builder farm performance

As described in 2.5, each SFI application is provided with a GbE link to the Datacollection network and a second to the BackEnd network. The design building rate is 3.5 kHz and the design event size is 1.5 MB, thus the Event Builder farm has to cope with a total bandwidth of 5.5 GB/s. Each SFI application is able to concurrently saturate the input and the output link, working at an effective throughput of about 105 MB/s. With 96 SFI applications deployed the maximum achievable bandwidth is therefore about 10 GB/s, twice the design value. This allows flexibility during trigger commissioning and guarantees an operational margin in case of temporary spikes in the trigger rate during the data-taking runs.

The maximum throughput of the Event Builder farm has been measured as a function of the number of building applications, for up to 93 SFI applications and an event size of 1.5 MB. The results are shown in figure 68. For each entry in the plot, the total throughput and the statistical error have been computed exploiting the information published in the Information Service (IS) for about 10 minutes. The slope of the line connecting the points corresponds to about 105 MB/s.

3.3 SFO performance

After installation of the new data storage farm during the spring of 2010 extensive tests have been performed to verify the expected performance characteristics, using 27 XPU racks configured as Event Filter, 90 SFI applications and 6 SFO applications. The simulated events used had an average

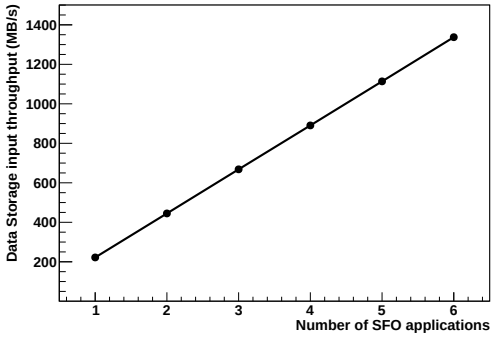


Figure 69. Maximum input data rate of the data storage farm as a function of the number of SFO applications.

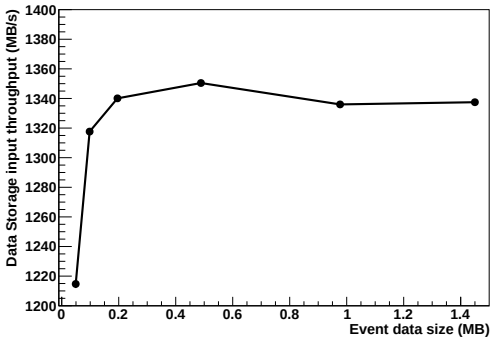


Figure 70. Data Storage input bandwidth as a function of the event size.

size of 1.5 MB. For each measurement the total throughput and the statistical error have been computed on the basis of information published in IS for about 10 minutes.

Each SFO node connects via 2 GbE links to the BackEnd network as described in 2.8.1, and each is running one SFO application. Each node can fully utilize the bandwidth of the input links, writing data files at a maximum achievable throughput of about 220 MB/s.

As shown in figure 69, the maximum input data rate with a data storage farm composed of 6 nodes is about 1.3 GB/s, much higher than the design working point of 300 MB/s. The maximum input data rate scales linearly with the number of SFO nodes, each contributing about 220 MB/s, as expected. As for the Event Builder farm, the possibility to write data files at more than twice the design speed is vital during trigger commissioning. The extra rate handling capability has been exploited during the first year of data taking characterized by a constant increase of luminosity and in particular during special runs for measuring the luminosity.

The size of the event data the TDAQ system has to deal with can vary between $O(100)$ kB and $O(1)$ MB. The normal event data size is somewhat smaller than the design size of 1.5 MB, the size of events generated by calibration triggers can be significantly smaller, especially during dedicated runs with a reduced number of sub-detectors, as performed during LHC inter-fill periods. The maximum data rate that the Data Storage farm can handle has been measured for event sizes varying from 50 kB to 1.5 MB in view of this, with 24 XPU racks configured as Event Filter and 90 emulated SFIs. Information on throughput and event size published in IS were used for producing the plot of figure 70. Approximately full utilization of the available input bandwidth of the storage farm is observed for events with a size of 100 kB or larger.

The SFO farm is composed of storage servers with 8 physical cores each. In order to benefit from the multi-core architecture and from hyper-threading, multi-threaded software must be used. The SFO application is multi-threaded, but a single thread, the "main thread", uses 95% of the available CPU power if the SFO application runs with full utilization of the input bandwidth and performs full event stripping, a procedure described in 2.6.2. The CPU load of the event stripping routine has been monitored, the results are shown in the left plot of figure 71. The CPU load varies between 10% and 20%, depending on the amount of event data stripped. The calculation of checksums consumes most of the CPU time. However, decoupling the event stripping from the

2016 JINST 11 P06008

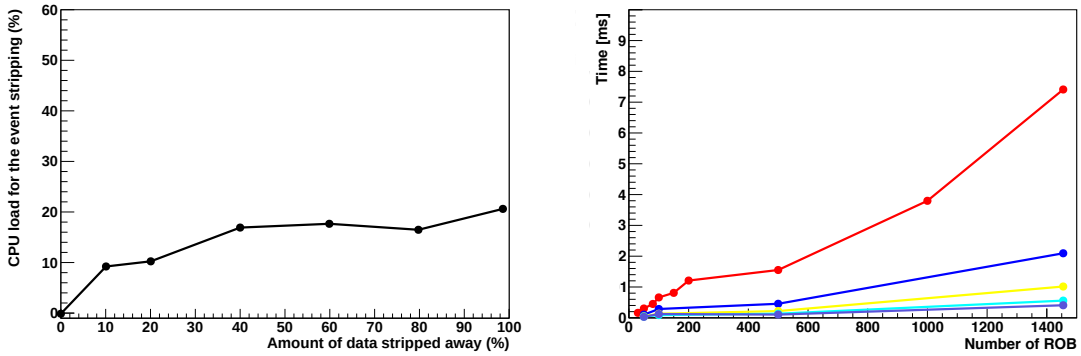


Figure 71. *Left*: contribution of the event stripping routine to the CPU load of the SFO application as a function of the fraction of the data stripped. *Right*: time needed to perform event stripping as a function of the size of the event data stripped, expressed in number of ROBs. The upper (red) line is for the actual configuration, exploiting a single thread; the lower (blue, yellow, turquoise and violet) lines correspond to using 2, 4, 8 or 12 threads, respectively.

main thread would allow a modest further improvement of the performance of the SFO application. The benefit of performing event stripping on multiple cores has been evaluated: the time spent on event stripping has been measured for different choices of the number of threads, using the Intel Threading Building Blocks library [163], the results are shown in the right plot of figure 71.

3.4 Cosmic ray data taking

Prior to the first collisions at the LHC, the whole ATLAS detector system including the TDAQ system was tested in several long cosmic ray data-taking periods. The trigger rates during cosmic ray running are low compared to those of collision runs, with about 1500 Hz of the lowest threshold L1 muon triggers, a few Hz of calorimeter triggers and only about 10 Hz of cosmic rays passing through the inner detector. It still proved very valuable for testing almost all of the TDAQ system as the cosmic ray runs required the full control and configuration system to run and also enabled the monitoring systems to be extensively tested. Besides testing the DAQ/HLT software, much experience was gained from the integration of the various detectors systems and the many possible failure modes that only show up with real hardware. To test the system behavior under high trigger rates, 10–75 kHz of random triggers were often added on top of cosmic ray triggers and then rejected immediately by L2. The cosmic ray periods were also the first opportunity to have the TDAQ system operated by regular operators rather than system experts and to run the full system for extended time periods. Some individual runs lasted more than 24 hours and helped to find memory leaks, etc. Based on the experience gained during the cosmic ray runs, many problems were fixed, new required features implemented and operational procedures established.

The cosmic ray data were not only used for testing, but also for detector commissioning, calibration and alignment [164–167]. To maximize the number of recorded tracks in the inner detector for detector studies, special HLT triggers were implemented to select events with one or more tracks in the inner detector running at the full 1500 Hz of muon L1 triggers. This was the first main use of the HLT to actively select events for ATLAS.

2016 JINST 11 P06008

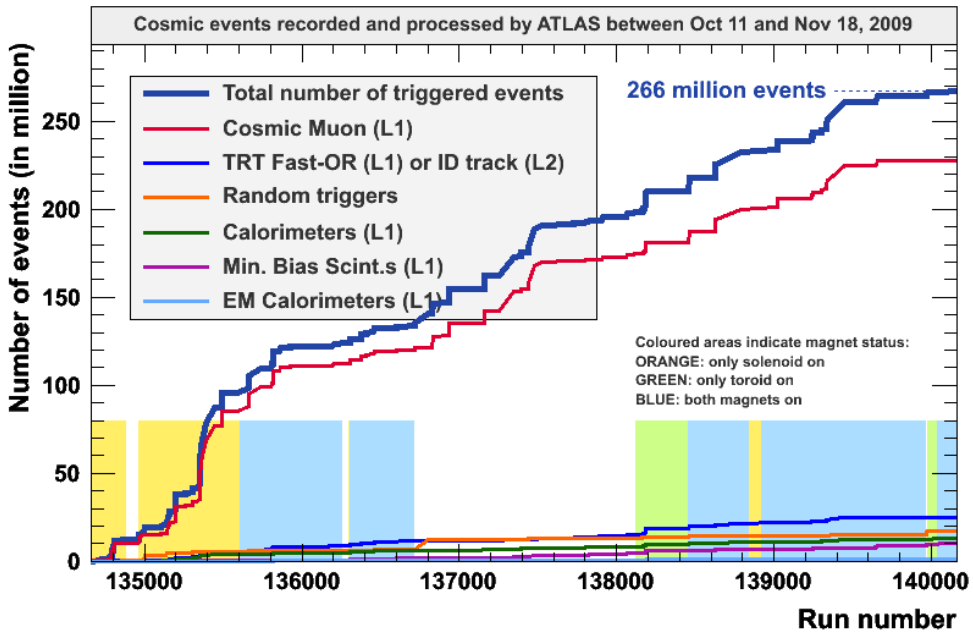


Figure 72. Size and composition of the event sample recorded during five weeks just prior to first collision data taking. The lines in the plot are ordered as implied by the legend, the blocks indicate that the solenoid and/or the toroid was switched on.

In total more than 500 million events were recorded during three combined cosmic data-taking periods in 2008/2009, each period lasting between two and six weeks, see figure 72 for the size and the composition of the event sample obtained during the last run. Additional cosmic ray data has continued to be recorded during longer periods without beams as these have been found to be useful for detector testing and alignment.

3.5 pp collision data taking

In this section plots with results on rates, fragment sizes, data collection times and trigger processing times, as observed during runs in summer and autumn 2011 for proton-proton collisions at 7 TeV are presented.

The three trigger rates (L1, L2 and EF output rate) during a run in October 2011 are shown as a function of time in figure 73. The steps in the rates result from real-time changes of HLT prescale factors (2.9.2), which are necessary to keep the final data recording rate roughly constant. The EF network traffic history for the same run is shown in figure 74. At the start of run the EB as well as the SFO traffic are well above the original design values of ~5 GB/s and ~300 MB/s respectively. During the run they are kept above 4 GB/s and 600 MB/s by changing prescale values for some streams. For stable running conditions, the event building time requires a few tens of milliseconds as shown in the left of figure 75.

The long processing time tail is more evident at the start of run when the EB network is close to saturation. The time distribution is consistent with the event size distribution as shown in the right plot of figure 75. It exhibits three populations: small calibration events, physics events and events for which more detailed data is output by the LAr calorimeter RODs. As expected the event size decreases slightly as the luminosity falls during a run.

2016 JINST 11 P06008

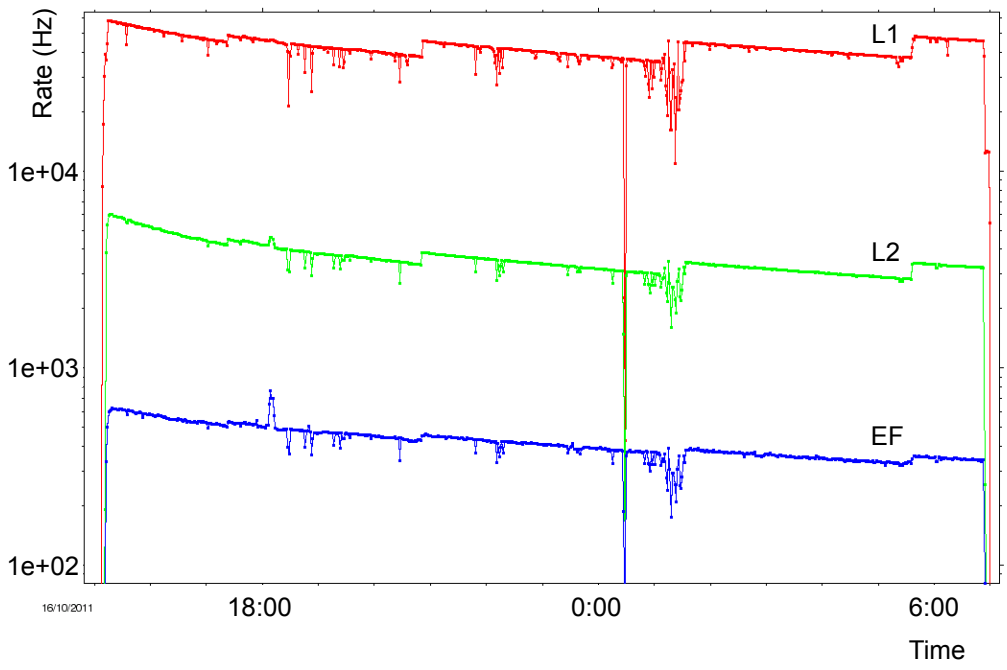


Figure 73. L1, L2 and EF trigger rates for a run with a peak luminosity of $3.5 \cdot 10^{33}$ cm^{-2} s^{-1} (16-17 October 2011, run 191190). Four major prescale changes are clearly visible (at about 5.30 pm, 9.00 pm, 1.30 am and 5.30 am). The peak at around 6 pm, mainly visible in the EF rate, is caused by a trigger noise spike. After midnight the trigger was temporarily halted because of a sub-detector readout problem: the subsequent rate oscillations are induced by the attempts to recover the faulty module.

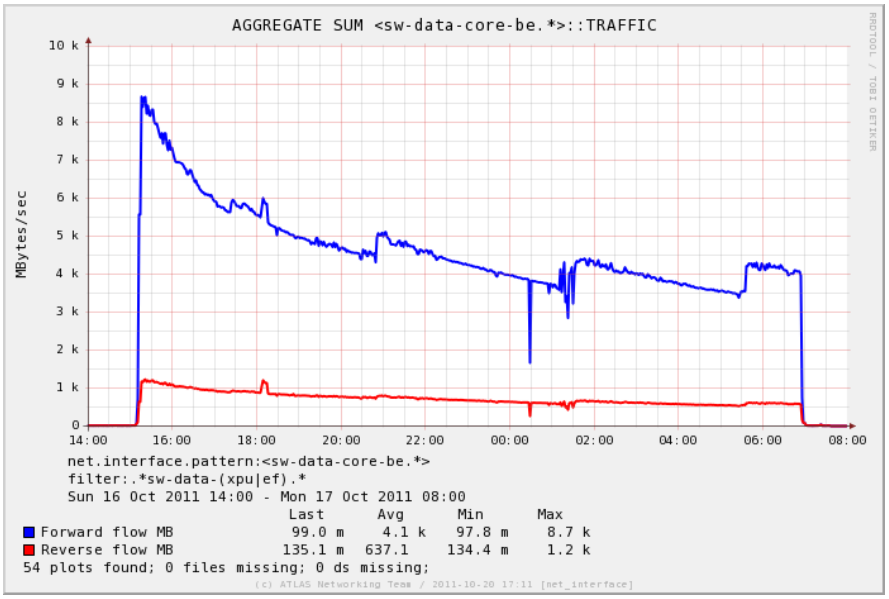


Figure 74. EF network traffic during a run with a peak luminosity of $3.5 \cdot 10^{33}$ cm^{-2} s^{-1}. Upper (blue) line: data traffic from EB to EF. Lower (red) line: data traffic from EF to data storage system (nodes running the SFO application).

2016 JINST 11 P06008

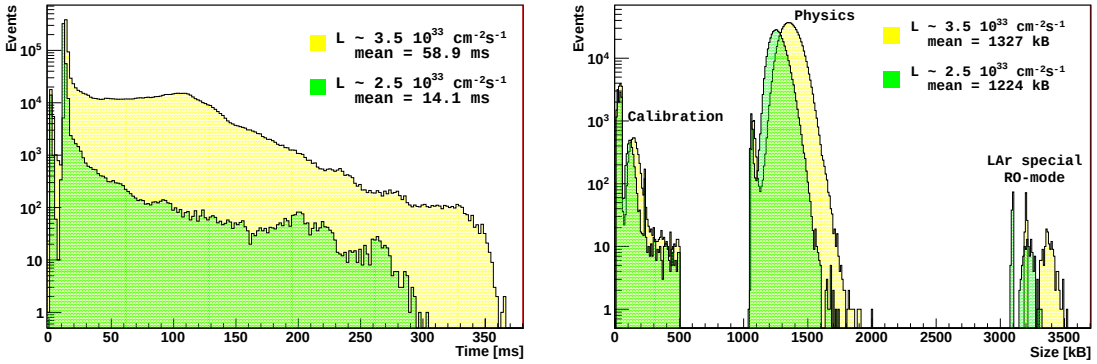


Figure 75. Left: event building time distribution. Right: event size distribution. Data was obtained from 100 s of stable running at two different luminosities (16-17 October 2011, run 191190). Trigger prescales change with the luminosity. As can be seen, calibration triggers can occur also during normal physics runs.

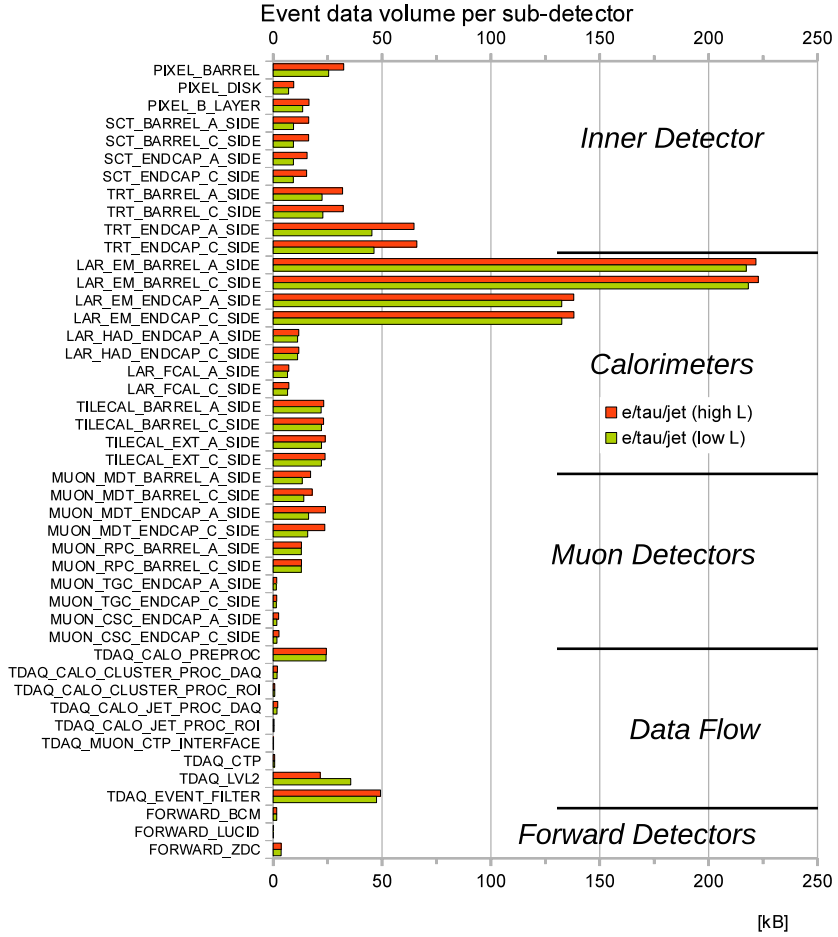


Figure 76. Contribution of each TTC partition to the event size for e/γ, τ and jet triggers, for a period at the begin of a run (red bars, high L) with a peak luminosity of $3.5 \cdot 10^{33} cm^{-2} s^{-1}$ and about 16.7 interactions per bunch-crossing and for a period at the end of the same run (green bars, low L) with an about 50% lower instantaneous luminosity. The size of the L2 result (TDAQ_LVL2) increases slightly near the end of the run as additional selection chains are enabled. The total event size is about 1.3 MB.

2016 JINST 11 P06008

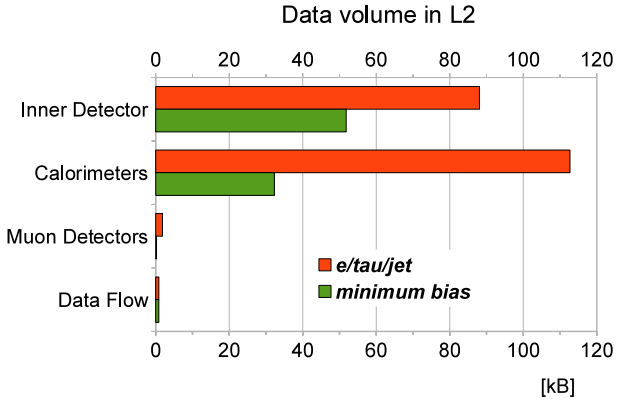


Figure 77. Average sizes of event data retrieved by L2. The data from the RoI builder pushed to the L2 system is labeled with "Data Flow". Typically about 86 kB of data are retrieved by L2 for minimum bias events, while for events with more complex signatures like jets or τ particles somewhat more than 200 kB of data are used for a decision. The data are from a run with a peak luminosity of $1.12 \cdot 10^{33} cm^{-2} s^{-1}$ and an average of 6.8 interactions per bunch-crossing.

The average amount of data per event and per TTC partition for collision data is shown in figure 76. Overall the calorimeters provide the largest contribution, about 1.3 MB, to the total event size.

The sizes of the data retrieved on average per event by L2 from the ROS for the calorimeters, the inner detector, the muon detector and the first-level trigger for minimum bias events and for events with e/γ and jet triggers are shown in figure 77. The retrieval rate is the input rate of L2, which is equal to the L1 accept rate and is considerably higher than the event building rate as can be seen from figure 73. According to trigger type, between 80 to 200 kB of the total event data are used for a L2 decision, of which about 1 kB of data is pushed to the L2PU via the RoI Builder and the L2SV that assigned the event to it.

The peak retrieval rate per ROS PC for data retrieved by L2 and for two different luminosities is shown in figure 78, and the peak data rate in figure 79. It can be seen that the LAr endcap calorimeter and the Tile calorimeter ROS PCs have to support the highest rates in terms of retrieval frequency as well as data volume.

The rate at which the L2 algorithms request data from the ROS PCs is shown in figure 80. Several algorithms may request the same data. To minimize the overheads associated with ROS data transfers, data retrieved from the ROS PCs is cached (2.4.3). Figure 80 shows that this results in a considerable reduction of the rate with which data is retrieved from the ROS PCs with respect to the rate with which data is requested from the cache.

L2 algorithms request data with the granularity of ROBs. The L2 Data Collector sorts the list of ROBs per ROS PC to produce the ROS data requests. There are typically 12 ROBs per ROS PC, see table 1. Figure 81 shows the average number of ROBs per ROS PC per request for which either the data is requested from the cache or retrieved from the ROS PCs. The average number of ROBs per request provide an upper limit for the average actually retrieved, since less ROBs will be retrieved if data from some of the ROBs are already cached. The different characteristics of the requests and the retrievals can be attributed to the different trigger menus deployed.

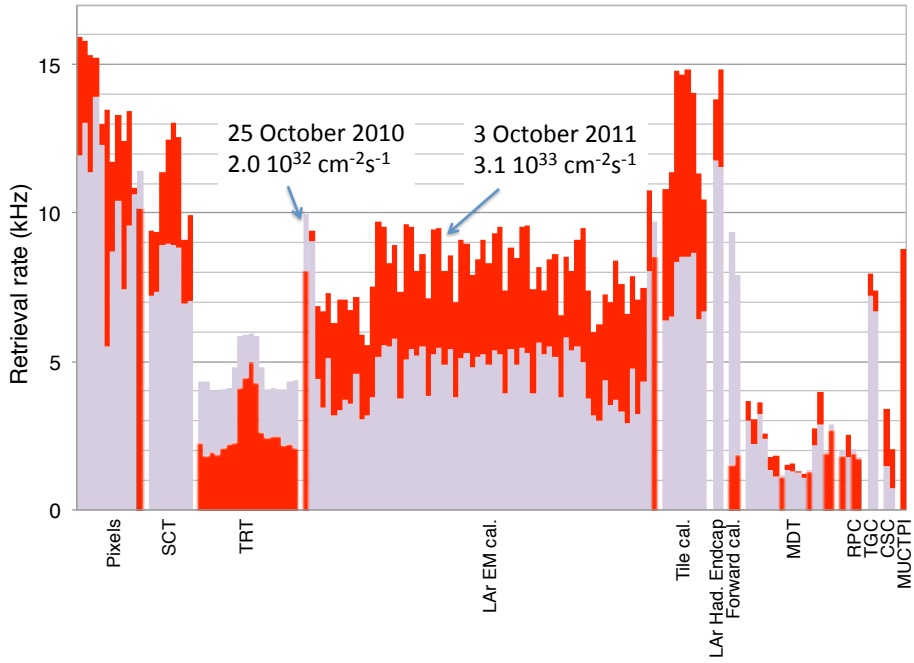


Figure 78. Peak retrieval rate per ROS PC of data retrieved by L2 for runs of 25 October 2010 and 3 October 2011 at the instantaneous luminosities indicated. The empty bins serve to separate the entries for different sub-detectors and are not associated with ROS PCs.

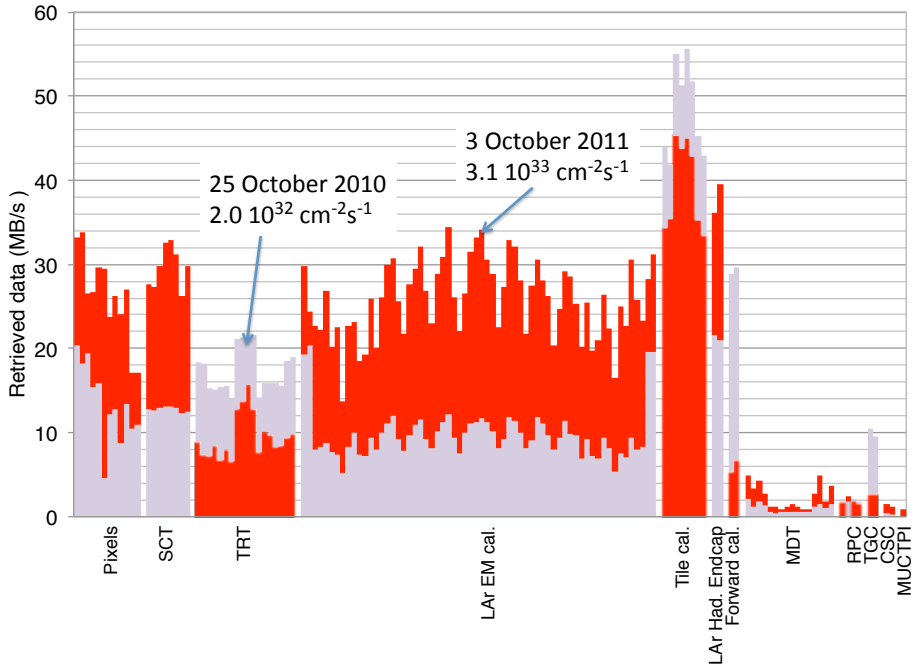


Figure 79. Peak data rate per ROS PC of data retrieved by L2 for runs of 25 October 2010 and 3 October 2011 at the instantaneous luminosities indicated.

2016 JINST 11 P06008

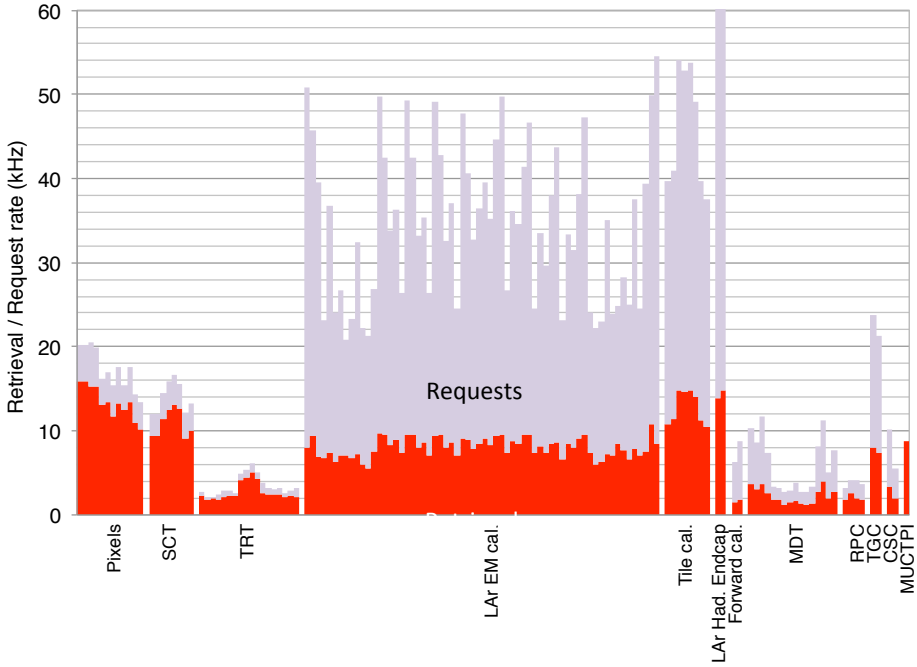


Figure 80. Peak L2 data request and retrieval rate per ROS PC for the run of 25 October 2011 (luminosity: 3.1 10^{33} cm^{-2} s^{-1}), showing the effect of caching. The peak for the request rate for the LAr hadronic endcap ROS PCs extends to about 121 kHz.

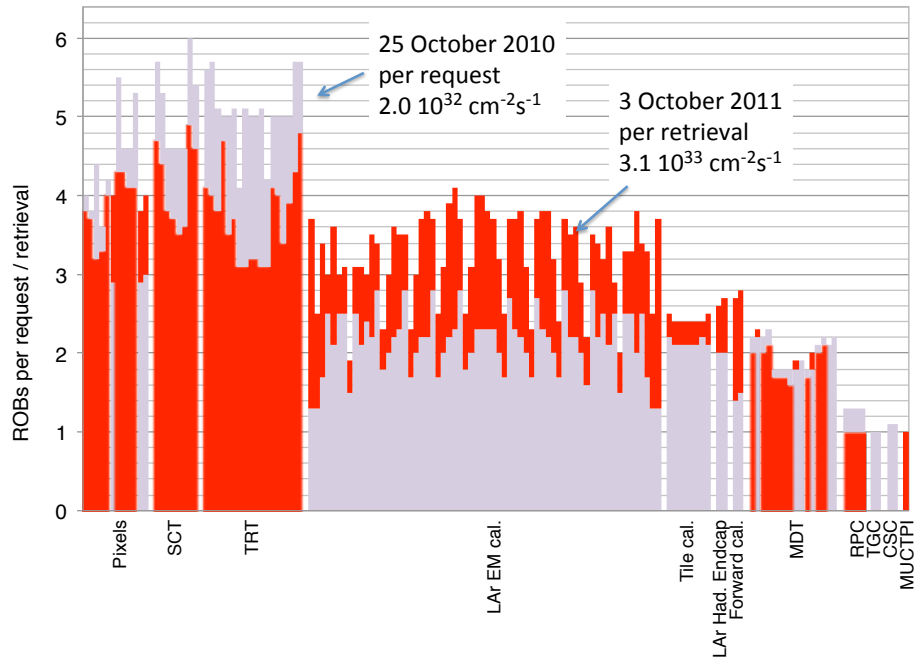


Figure 81. L2 algorithms request data from the ROS PCs with the granularity of a ROB. This plot shows for each ROS PC the average number of ROBs, for the run of October 2010 as requested by the algorithms, and for the run of October 2011 as retrieved from the ROS PCs. L2 E_T^{miss} requests were not yet implemented.

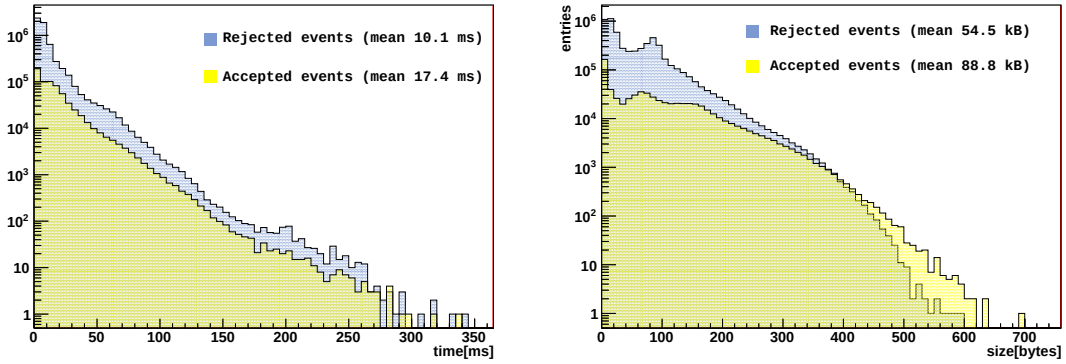


Figure 82. Left: L2 collection time distribution. Right: amount of data collected by L2. The distributions are for triggers occurring during 100 s of stable running at $3.5 \cdot 10^{33}$ cm^{-2}/s^{-1} (run 191190, 16-17 October 2011).

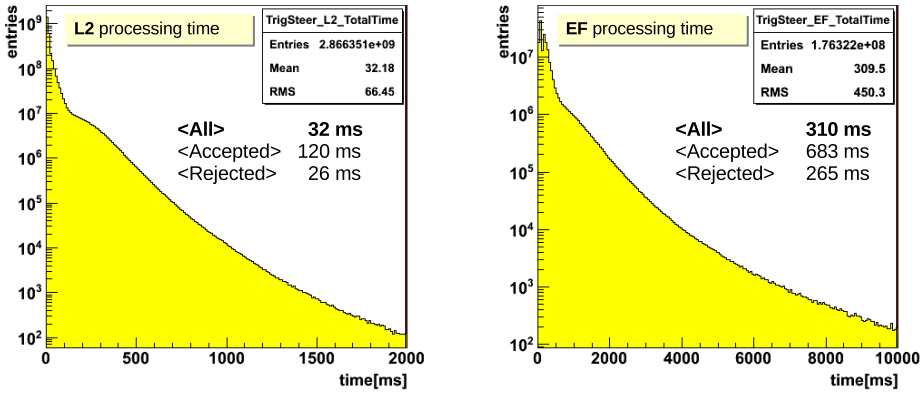


Figure 83. L2 and EF event processing times. The L2 processing time includes the time needed for data collection. The data are from a run with a peak luminosity of $1.12 \cdot 10^{33} cm^{-2} s^{-1}$ and an average of 6.8 interactions per bunch-crossing.

The distribution of the time needed to collect the data required for L2 processing and the distributions of the sizes of accepted and rejected events are shown in figure 82, distributions of the L2 and EF event processing times in figure 83. As can be seen accepting events takes longer than rejecting events, reflecting the sequential processing of the HLT algorithms.

The processing time per signature relative to the total processing time is shown in figure 84. It is dominated by track reconstruction in the inner detector for B physics triggers.

The memory usage by L2PUs and EFPUs over time is illustrated in figure 85. It is just below 1.2 GB per process. It should be noted that each machine is capable of running 8 or 12 processes (two CPUs, four or six cores per CPU) within a memory budget of 16 or 24 GB.

Figure 86 shows the rate distributions for L2, EB and EF. The EB, consisting of homogeneous hardware, is perfectly balanced: all the SFIs are working at the same rate. Although the HLT hardware is heterogeneous (two types of CPU are deployed), the L2 and EF farms also exhibit a good load balancing: the widths of both distributions are significantly less than 10%. Figure 87 shows the CPU utilization by the L2, EB and EF farms during the run of 16-17 October 2011. A safe margin is evident for all the sub-systems. As expected the CPU utilization decreases with the

2016 JINST 11 P06008

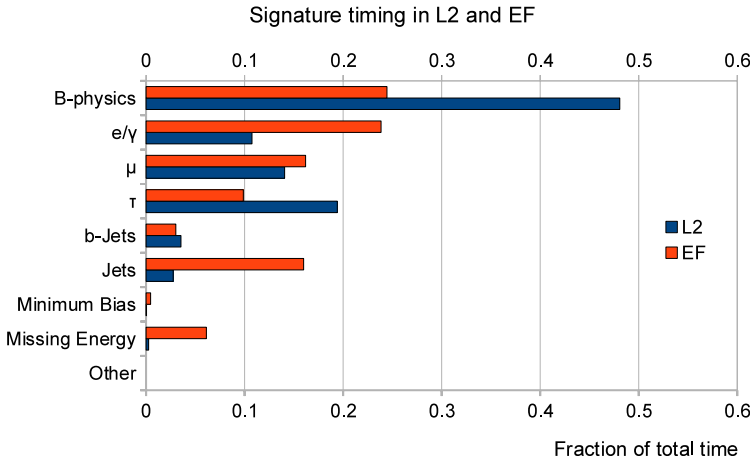


Figure 84. Time spent per event signature relative to the total processing time at L2 and EF in a run with a peak luminosity of $1.12 \cdot 10^{33} cm^{-2} s^{-1}$ and an average of 6.8 interactions per bunch-crossing.

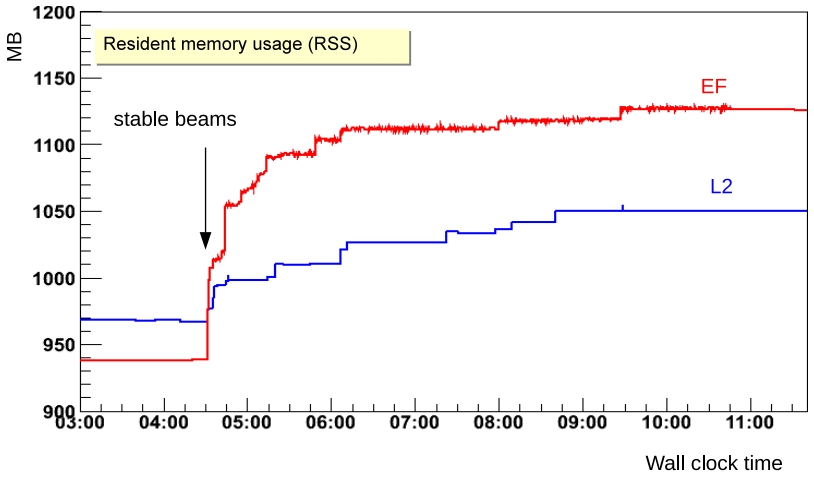


Figure 85. Memory usage by L2PUs and EFPUs over time. The arrow indicates the declaration of stable beams and the start of data taking.

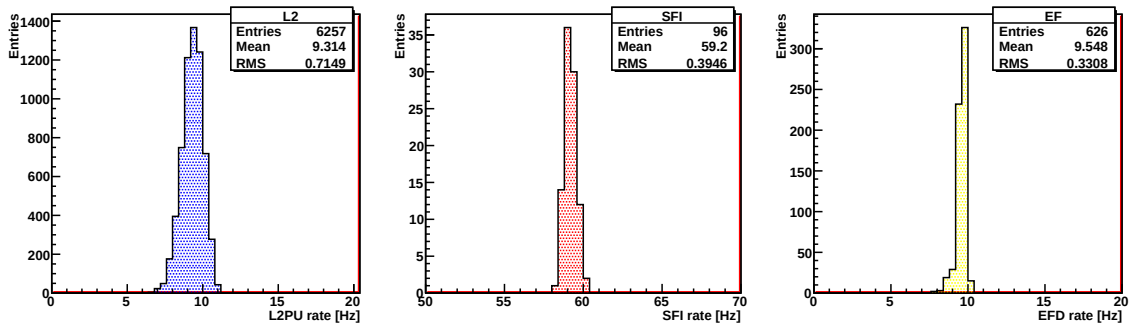


Figure 86. Rate distributions for the L2PU (left), SFI (centre) and EFD (right) applications. The EFD applications supplied events to about 6400 EFPUs. Data was taken at a luminosity of $3.5 \cdot 10^{33}$ cm^{-2} s^{-1} (run 191190, 16-17 October 2011).

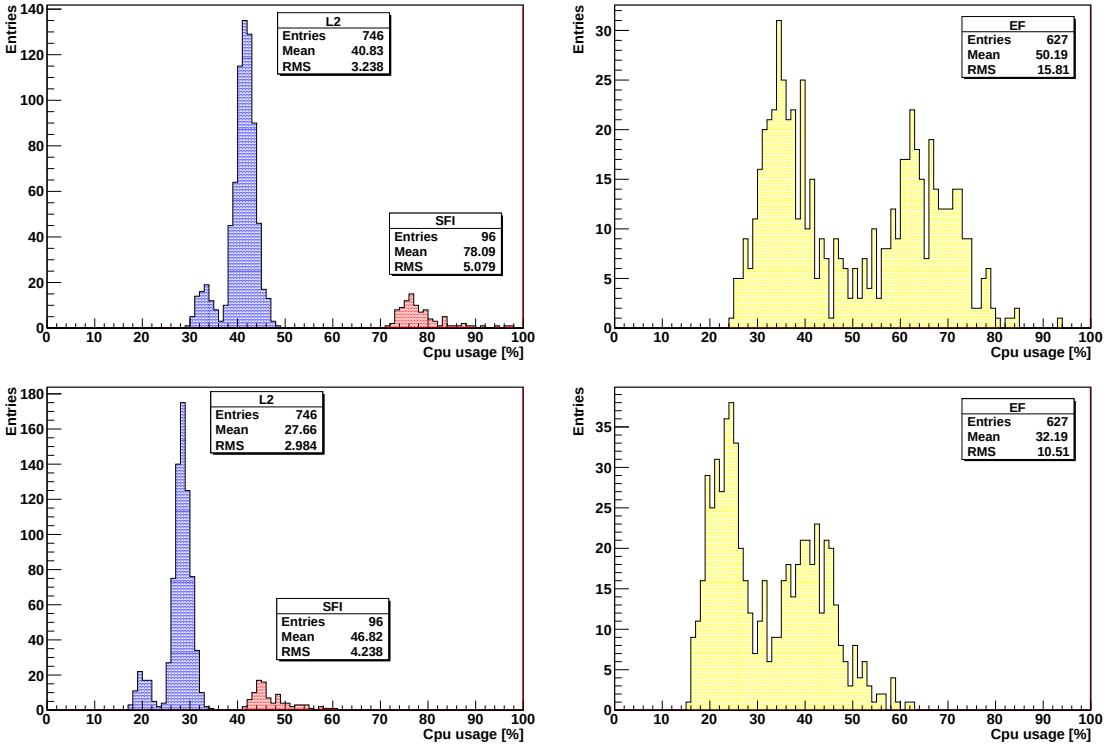


Figure 87. Distributions of CPU utilization per node for L2, EB (left) and EF (right). The upper histograms were taken at the start of of a fill ($L = 3.5 \cdot 10^{33}$ cm^{-2} s^{-1}), the lower ones after 4 hours of running ($L = 2.5 \cdot 10^{33}$ cm^{-2} s^{-1}). The distributions are wider for EF than for L2 because the time sampling interval of 10 s used for determining the CPU utilization was smaller than the EF processing time scale.

luminosity. The events are assigned to the processing nodes according to a fair scheduling policy. Therefore all the nodes run at about the same rate if the system is not CPU limited, but the CPU utilization of each node depends on its computing power. This causes the double peaks in figure 87: the first peak was populated by the newer and more powerful nodes, which represented respectively 10% and 50% of the processing resources of the L2 and EF farms.

Figure 88 shows the data acquisition efficiency for a sequence of LHC physics fills. For each fill, the top plot shows three different efficiency values, respectively defined by:

- "DAQ efficiency" (left, blue): fraction of the time with beam when the data taking is enabled (i.e. running and no dead time).
- Fraction of the time with stable beam when the data taking is enabled and the detector is in full physics mode[27] (middle, red).
- "Physics DAQ efficiency" (right, orange): as the previous item, but excluding the stable beam time lost due to the beam dump handshake. This procedure is in fact known to artificially introduce inefficiency since the detector must necessarily be switched off before the actual beam dump.

[27]For safety reasons, several sub-detectors can be fully powered only if beams are declared stable.

2016 JINST 11 P06008

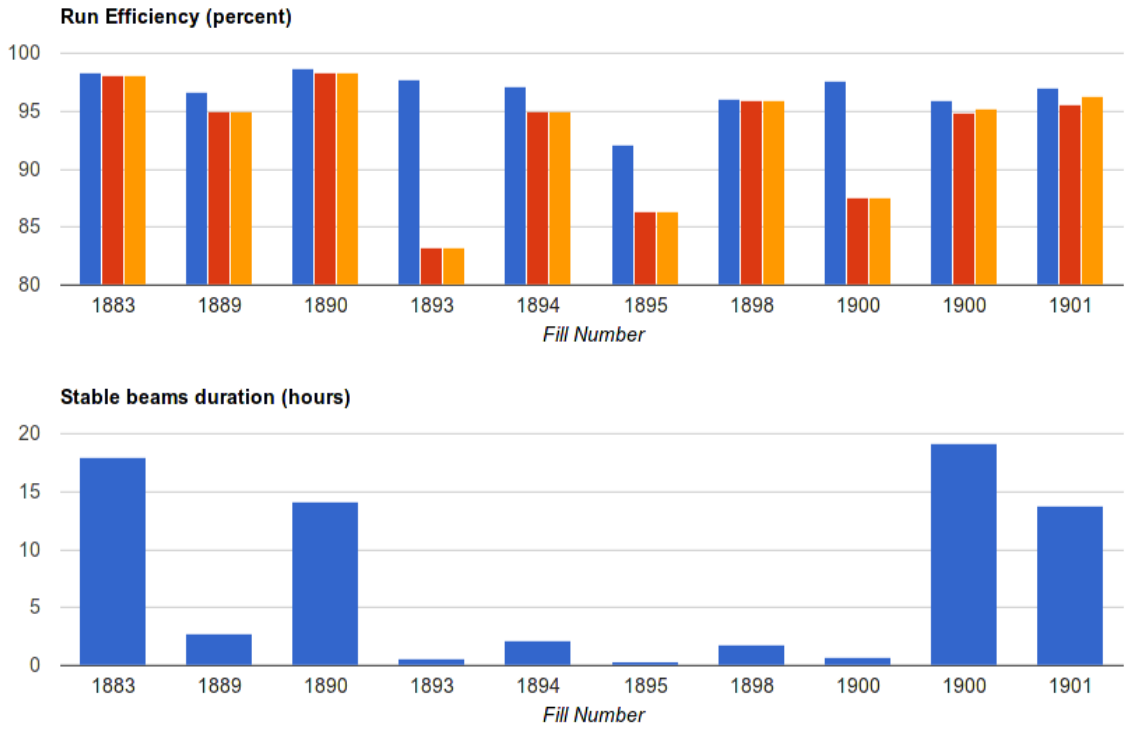


Figure 88. Data acquisition efficiency and stable beam duration for the period 20 June 2011 to 30 June 2011 during which the LHC stably operated at peak luminosity of roughly $1.1 \cdot 10^{33}$ cm^{-2} s^{-1}. The different efficiencies are defined in the text.

The bottom plot shows the stable beam duration for each fill. For fills lasting more than a few hours, normally the achieved physics DAQ efficiency is 95% or more. For shorter fills, the time necessary to switch on the detector after the beams are declared stable is not negligible, the physics DAQ efficiency therefore is significantly affected. However, the DAQ efficiency is not affected, since it does not depend on the detector state.

4 Discussion of design and technology choices

4.1 The role of modeling

Estimation of rates and identification of possible bottlenecks are essential for the design of a trigger and DAQ system. For this purpose a static and a dynamic model were developed and used, the so-called paper model and the computer model.

4.1.1 The paper model

First work on the paper model was, as the name of the model suggests, in the form of simple calculations with the help of a pocket calculator on a piece of paper [168]. Then spreadsheets were used for more involved calculations [169–172]. The RoI driven L2 trigger causes the required calculations to be less straightforward than those needed for a conventional system, as the processing by L2 has a direct impact on the patterns of requests sent to the ROS PCs. These patterns depend on

the nature of processing in the different processing steps, the acceptance factors of the processing steps, the mapping of the detector on the ROS PCs and the distributions of the RoIs of different type. Over time the available information became more detailed and averages could be replaced by tables, at the same time making the use of a spreadsheet problematic. A small C++ program provided a solution and made it straightforward for example to study the effect of different mappings of the detector on ROS PC request rates. It should be noted that the positions of the RoIs are assumed to be uniformly distributed in η-ϕ space, with minimum and maximum values of η determined by the sub-detector system concerned. The L1 trigger assigns each RoI to a point on a grid of possible positions. For the different type of RoIs (muon, electron/photon, τ and jet) there are different grids. In the model for each type of RoI an area in η-ϕ space is associated with each grid point. If the grid is rectangular this area is equal to the area of one cell of the grid. The size of the area determines for each position on the grid the relative RoI rate for the type of RoI considered. The L2 trigger requests data from the ROS PCs on the basis of the positions of the RoIs and their sizes, where the size of a RoI is chosen by the L2 trigger on the basis of the type of RoI. In the paper model the relative RoI rate is calculated for each type of RoI for each possible RoI position, then from the size of each type of RoI and from the mapping of the detector on the ROBs the relative request rate for that type of RoI is calculated for each ROB and also for each ROS PC. Next the rate for each item in the L1 trigger menu is determined on the basis of the probability specified for each item. For each of the L1 menu items the L2 trigger chain needs to be specified, and for each step: (i) the number and type of RoIs involved and for each RoI the sub-detector(s) from which data will be requested, (ii) the acceptance factor, (iii) the average processing time (for determining the required L2 processing resources). On the basis of this specification the total RoI request rate per ROB and per ROS PC can be calculated and also for each ROS PC the average number of ROBs from which data is requested per RoI request. On the basis of fragment sizes specified as input for the model the bandwidth requirements can also be determined.

The results of paper modeling provided guidance for the design of the TDAQ system, in first instance for studying the relative merits in terms of throughput of different approaches of the design of L2 and for the Technical Design Report of 2003 [30], in particular with respect to grouping of ROBs per ROS PC, rate handling capability and output bandwidth per ROS PC, network throughput and rate handling capability of the L2 farm processors.

With the availability of results from data taking the paper model could have been further refined, in particular the uniform distribution of the RoIs in $\eta - \phi$ space could have been replaced by more realistic distributions. However, the cost monitoring tools (2.13) provide detailed results on the request patterns generated by L2 and also on the CPU time spent by the various algorithms of each trigger chain. For this purpose monitoring data is acquired in the L2 system during data taking, so all events handled by L2 are taken into account, not only events accepted by L2. The detailed results for each trigger chain make it possible to predict the effect on the rates of a modification of the L1 trigger menu and/or L2 trigger menu. The results therefore allow the optimization of the menus as a function of the luminosity taking into account the maximum rates that can be handled and the physics to be studied. The predictions of the rates on the basis of the cost monitoring results will be more accurate than those of the paper model. Furthermore there is no need to obtain input required for the paper model from an analysis of the data and to specify the complicated trigger menus and for each trigger item the details of processing plus acceptance factors for each processing step. Therefore no further usage of the paper model is foreseen and further development of it is no longer planned.

2016 JINST 11 P06008

4.1.2 The computer model

The computer model consists of a simulation of the behavior with respect to data flow of components of the DAQ system (ROBs, ROS, DataCollection network, nodes running the DFM and the event building nodes) and the L2 trigger (the L2 farm and the nodes running the L2 supervisor and L2 Result Handler applications). It allows the dynamic behavior of the system to be studied, in particular the effect of queueing of messages and of load balancing of the L2 trigger processors and the SFIs (i.e. the event building nodes). The type of simulation is know as discrete-event simulation. The simulation program maintains a queue of time ordered "events", i.e. objects that contain information on the (simulated) time at which they should occur, together with other information. This can be information on the event itself and/or on an object that will be affected by the occurrence of the event. The simulation program fetches events one-by-one from the front of the queue and processes each event. This could result in the generation of new events that should occur at a later time and that are stored in the queue, where the position in the queue is determined by the time at which the event should occur. In this way a complex system with many parallel processes can be simulated on a sequential computer. The first models were developed with a special simulation language, MODSIM-II [173–176]. Later work was done using the Ptolemy environment [177–179], with special attention to models of networking technology. In parallel also C++ with direct implementation of the support required for discrete-event simulation was used [180–182].

The computer model implemented in C++ made it possible to tune the discrete-event mechanism to obtain optimum performance of the simulation program and therefore allowing a detailed model of the DAQ and L2 system to be run. Another feature of the C++ program is the way in which the internal events are handled, which made it possible to develop complex models. Each object representing an event only contains a pointer to the object that caused the event and the time at which it should occur, but no information on the nature of the event. The object fetching events from the event list invokes with the help of the pointer the EventHandler method of the object that caused the event. The latter object has a set of states associated with it and can only be in one of the states. Submitting an event to the event list, invocation of the EventHandler method or interaction with another object can cause a state transition. A typical example is the simulation of a process that takes some time to complete and then has an effect on another object. This can be modeled with the help of two states, e.g. labeled with *WAITING* and *ACTIVE*. Typically first a transition from *WAITING* to *ACTIVE* will occur because another object will invoke a method that causes this transition. Then an event will be submitted for a time in the future corresponding to the time that is needed to complete the process and the method returns. Once the event is fetched from the event list the EventHandler method of the object will cause a transition from the *ACTIVE* state to the *WAITING* state and invoke a method of the object that was waiting for the processing to complete, which in turn will cause a state transition in that object. Complex systems and behavior can be simulated in this way using interacting objects operating on the basis of simple state machines without any special support required other than provided by an object-oriented language like C++ and a good programming and debugging environment.

The computer model makes use of the relative RoI rates calculated for the paper model for each type of RoI for each possible RoI position. Events ("physics events", not to be confused with the simulation events used in the simulation) are generated, where for each event the association with an L1 trigger menu item is determined on the basis of the probabilities of these items. The menu

item specifies the number and type of the RoIs. Their positions are determined for each event on the basis of the relative RoI rates, which are probabilities. For each event is also determined, on the basis of the acceptance factors, which steps of the L2 trigger chain will be executed. Then the event is advanced, at the simulation time at which it should occur, to one of the simulated L2SVs and the fragments are sent in the simulation to the ROBs. The ROBs and therefore also the ROS PCs from which data need to be requested are determined from the positions of the RoIs and using the same input data as the paper model. Assignment of an event to an L2 processor, sending of requests to ROS PCs via the network through network switches and of responses by the ROS PCs to the L2 processor etc. all proceed in the computer model in the same way as in the system simulated. In addition to the input required for the paper model the computer model also needs processing times, task switching times and transfer speeds to be specified. The average message rates and bandwidth utilization should be equal within statistical fluctuations to those obtained with the paper model if the utilization of the processing and network resources in the simulation is not too high and if the model is run long enough so that the effect of the "cold start" at the beginning of the simulation can be neglected. With the C++ program it has been possible to simulate several 100,000's of L1 accepts per hour, i.e. a few seconds of data taking per hour. Typically with a simulation run of a few hours accurate reproduction of the paper model results was possible. A detailed comparison of the results proved to be useful for checking and improving both models.

In addition to the results of paper modeling, results of computer modeling provided support for the design of the TDAQ system, as described in the Technical Design Report. An overview is also presented in ref. [183]. The results show clearly for example that round-robin assignment of events to the L2 processors results in a long tail for the L2 latency at a moderate level of utilization of the L2 processors (80%) and that this behavior can be improved by using a more optimal assignment strategy. This also results in a reduction of queueing in switches.

After submission of the Technical Design Report the technology models implemented in Ptolemy were ported to the model implemented in C++. The latter was adapted to obtain a more faithful model of the system implemented. In particular a model for VLANs, as used in the real system, was incorporated.

For the computer model the same problems with the input occur as for the paper model: the distribution in η-ϕ space of the RoIs need to be made more realistic and the specification of the L2 trigger chains is problematic. The cost monitoring tools cannot replace the computer model. However, the data recorded by cost monitoring are small events without any physics data, but with the information that is needed in the computer model, such as which ROBs were requested to supply data and the time at which the requests were sent, the amount of data received, the execution time for an algorithm, etc. Large quantities of these events are available. It was therefore planned to base the generation of "physics events" in the computer model on these events instead of using the method described earlier. This would solve the problems mentioned, first steps have been taken to implement this approach.

4.2 The boundary between sub-detector and TDAQ domains

A clear boundary at the interface between the sub-detector specific RODs and the DAQ system has been a requirement for the design of the DAQ system, in view of benefits with respect to their integration and management. This has been achieved by the early agreement to use the S-link protocol [17, 18] for the ReadOut Links (ROLs) and to make use of mezzanine boards, with a well-

defined 32-bit parallel interface, on the RODs for interfacing to the physical links. The latter made it possible to adapt to the latest technology available for the links at the time of mass production of the mezzanine cards, while for designs of the RODs the technology used for the links did not play a role.

The buffers in the ROBs could have been integrated into the RODs, but in particular for testing of the DAQ system this would have been problematic with respect to interactions with the ROBs. Using preloaded data in the ROS PCs or test generators built into the ROBINs the full DAQ system could be tested without the RODs required to be present or to be powered. This has proven to be a valuable approach, not only for commissioning, but also for validating new releases of the TDAQ software. However, the required output bandwidth of the RODs could have been decreased by forwarding requests directly to the RODs and outputting only the requested data. This would have necessitated random access to the event data in the RODs and also support for deleting event data, making the design of the RODs, the interfacing to the ROLs and also the testing procedures more complex. The risk of subtle design errors, in particular in the interfaces to the ROLs and therefore at the boundaries of systems implemented by different groups, would also have been larger.

4.3 ROS technology

The design of the current ROBINs evolved from earlier designs, see for example refs. [184–186]. Initially it was thought that the ROBs had to be housed in for example VME crates or crates with a PCI backplane. However, once server computers with a sufficient number of 64-bit PCI slots and with interfaces to GbE became available it became clear that these would provide a good and cost effective environment for housing the ROBs. At that time it was agreed that the ROBIN card should be developed as a 64-bit PCI card with a paged buffer memory as described in ref. [184], with an FPGA managing the data streams and with a PowerPC processor keeping track of the pages in use and the L1Ids of the fragments stored in these pages and handling requests for data and delete commands. The first prototypes contained two ROBs, i.e. two inputs for S-link connections and two buffer memories [187, 188]. The final version contains three ROBs as this was judged to provide a good balance between cost and performance. The on-board programmable processor has proven to be a valuable asset due to its ability to handle tasks that are not straightforward to implement in an FPGA, in particular keeping track of occupied and free memory pages. Furthermore control and monitoring are tasks that in practice require the flexibility that only can be provided by software, therefore also for executing these tasks a programmable processor is required. With the technology available at the time of the design of the ROBIN, which started in 2001, using the CPU of the PC as that processor did not seem to be feasible[28].

For the original ROS PC the performance is determined by the processing capacity of the CPU and not by the ROBINs. Although the original requirements for request rates and data throughput are satisfied it was found that the limitations of the ROS PCs could have a negative effect on the physics output of the experiment. Since the commissioning of the ROS PCs much more powerful CPUs and also faster memories have become available. After an evaluation of possible upgrades, including making use of the GbE interfaces of the ROBINs, it was decided early in 2011 to replace the mother-boards, CPUs and memories of all installed ROS PCs and spare ROS PCs piecemeal during technical

[28]Since the readout system needs to be extended for additional RODs (for the "Insertable B-Layer" [189], for other sub-detectors, for the L1 system and for the new track finder, the FTK [190]) a new ROBIN is being designed. The new design does not have an on-board processor, as the CPU of the ROS PC is now expected to be able to execute its tasks.

stops in 2011, as already mentioned in 3.1.2. The performance of the ROS PCs increased considerably and in some cases the rates are limited by the ROBINs rather than by the processing capacity of the CPU of the ROS PC or by the available output bandwidth of the two GbE links. Tests have shown that maximum performance is achieved with two of the four cores used, where the two should share the same L2 cache memory (the processor has two pairs of two cores, with each pair sharing one L2 cache memory). The original PCIe network interface cards have two unused ports, these could have also been used and would make a higher output bandwidth possible, as has been shown with test setup measurements (3.1.2), but would require more network infrastructure to be installed.

The ReadoutApplication, the program running on the ROS PC, receiving and forwarding requests to the ROBINs and forwarding data received from the ROBINs to the L2PU or SFI requesting the data, is multi-threaded. This is essential for achieving good performance of the standard ROS PC and also for exploiting the multi-core processors used for upgrading the ROS PCs. Further optimization should be possible, as indicated by the observation that maximum performance is obtained when using two out of four cores for the new processors for the ROS PCs. It has indeed been found that a different organization in terms of threads results in good performance achieved with all four cores used. However, the performance obtained is at about the same level as observed for two cores and as observed for the original version of the ReadoutApplication running on two cores, but at the cost of more power dissipation.

For a request received by the ROS PC a ROBIN will receive up to three requests associated with the same event as they are forwarded to the ROBs. Combining these requests provides another opportunity for optimising the performance.

The plugin architecture of the ReadoutApplication enables it to be configured in alternative ways. This has been found to be very useful for testing, where similar functionality is needed, provided by the application running in the standard configuration. Furthermore it has allowed reuse of a large fraction of the code for implementing the ROD Crate DAQ (RCD) application (2.3.5).

4.4 RoI driven L2 triggering

4.4.1 Motivation

The RoI based approach emerged in the early design stage of the ATLAS TDAQ system to address two major challenges: the desire to provide an efficient online rejection of high rate background events while maintaining unbiased efficiency for rare signals and the severe technological difficulties to handle the expected data volume and rates. It was therefore decided to reduce the trigger rate and data volume in three stages, as implemented in the current system. Furthermore, it was decided that the results of each stage would be used to seed the processing in the subsequent stage. Thus, the output of L1 (via the RoI Builder, see section 2.4.1), would provide details such as the identification of the L1 data channels, modules and threshold levels that led to the trigger decision. This information seeds the L2 selection, which refines the L1 decision using fine grained detector data and more elaborate algorithms than L1. Data from other sub-detectors not involved in the L1 decision, in particular from the inner detector, may also be used. Likewise, L2 and the EF pass details of their decision to the next stage. This approach was to offer several advantages: factorisation of the network in two parts (RoI data collection for L2, event building for the EF), a huge reduction of data network capacity (for a reduction of the trigger rate by L2 of ~30 using only

2-3% of the total event data needs to be transferred) and, for the selection software, a reduction in processing time by concentrating on RoI regions.

4.4.2 Historical background

Initial studies of the ATLAS TDAQ system started in the mid 1990s within the framework of CERN R&D projects [191] anticipating the approval of the LHC. One of the first proposed architectures for the L2 trigger consisted of dedicated processors ("Feature Extractors") which were connected to the readout of small detector segments thus partitioning the network in a number of smaller elements. An additional global network connected all Feature Extractors to a set of "Global Processors" which produced the L2 trigger decision based on combining the features obtained from the Feature Extractors [192]. Various technologies such as Transputers and Digital Signal Processors (DSPs) with their specific links [193] as well as VME based single board computers interconnected by other network technologies were investigated.

With the progress in network technology and the advance of cheap and increasingly powerful PCs (Moore's Law [194]) a radical solution based on more conventional components was proposed. The main ingredients of the L2 Pilot Project [195] were: an L2 architecture very similar to the current one, PCs running Linux or Microsoft Windows NT, three networks (Control, L2 data, EB data) and the use of C++ in a common software framework for the data flow (the "Reference Software"). Three different technologies were proposed for the networking: ATM [193], SCI [193, 196] and Fast Ethernet with custom protocols. The current Message Passing layer is a relic of the software layer introduced to provide an abstract software interface to these technologies. In the longer term, high-end rack mounted PC servers, Linux, (multi)Gigabit Ethernet and TCP/IP have been chosen as suitable technologies. This period culminated in the submission of the ATLAS High-Level Triggers, DAQ and DCS Technical Proposal (TP) in March 2000 [197].

4.4.3 Convergence

After the TP, the TDAQ architecture was to a very large extent already determined. The earlier software was re-engineered for the entire data flow and integrated with the Run Control software using C++ and targeted to PC farms with Linux as operating system. An important decision was to develop and test the trigger software in the offline environment. This had profound consequences for both the online and offline software. Large parts of the offline software, including its infrastructure, now had to run online. Likewise, offline software had to be adapted to allow interfacing to online services such as the event data source (acquired in real-time online and even piecemeal in L2), error reporting, monitoring, histogramming and database accesses. It also had to become aware of the Run Control states.

The L2 processors run an SMP Linux kernel to support multiple CPUs, each implementing multiple cores. Hence, L2 processes were designed to be multi-threaded allowing several events to be processed in separate worker threads. This was not supportable with the offline software. Consequently, in each L2PU application the trigger selection is now performed by a single worker thread. With the integration of the offline selection software it became possible to inject simulated detector data (into ROS PCs and L2SVs) to run the entire selection chain on multi-node TDAQ testbeds.

The work culminated in the submission of the ATLAS High-Level Trigger, Data Acquisition and Control Technical Design Report (TDR) in June 2003 [30].

2016 JINST 11 P06008

4.4.4 Status and outlook

The period after the submission of the TDR was followed by refinements of data flow (e.g. with respect to robustness, fault tolerance, multiple output streams, monitoring, access to databases) and selection software (e.g. with respect to trigger configuration, menus, calibration). The TDAQ hardware was installed and commissioned in test runs initially with Monte Carlo simulated data and eventually cosmic ray data.

The use of common selection software in online and offline has many advantages. It shortens the time and effort for testing, brings coherency in the updates, allows for consistent normalisation in the analyses and enlarges the pool of experts. The online and offline software produce identical results. Events that fail the online selection (because of possible glitches in some of the many thousands of processes) are saved in a debug stream and recovered in offline mode.

Results reported in 3.5 indicate that the reduction in data rates and volumes is as expected. Initially data requests to some ROS PCs were disproportionally high. This could be improved by optimizing the distribution of ROBs over ROS PCs. A good proportion of the L2 resources (time and bandwidth) are used for B-physics triggers. In fact, some earlier design studies included special hardware co-processors to speed up selection but this was abandoned. However, a new track finder built from dedicated hardware, the Fast TracK finder (FTK) [190], is at present being implemented. It will receive a copy of the data output by the RODs of the pixel and SCT sub-detectors and output parameters of track segments at the full L1 accept rate. It is foreseen that the output of the FTK will be sent to dedicated ROS PCs from where the data can be requested by L2.

Multiple constraints make it difficult to balance the processor resources devoted to L2 or EF: the same number of racks has to be assigned to each L2SV, each L2 or EF sub-farm has to have the same processing power and the EF sub-farms should have the same input bandwidth. In general modification of the configuration database is required to modify the working point. Architectural changes implemented during the shutdown of 2013-2014 allow a more gradual transition between the two HLT levels by executing both L2 and EF selection within the same process.

4.5 Data flow aspects

4.5.1 Push vs. pull architecture in the L2 trigger

Early proposals for constructing the L2 ATLAS trigger often considered a system built to a large extent of custom-made hardware (4.4.2).

The term push architecture was used for a data flow model where all event data in a Region of Interest (RoI) would be immediately sent to the processing unit that would deal with the event. While a fully functional push system was never implemented, it would be in today's terms the task of the L2 supervisor to inform the ROS what RoI data should be sent to which processor. Implementations differed in how the assignment was done, e.g. a round-robin assignment based on event numbers would not have even required a supervisor at all. The L2 algorithms considered at the time were rather simple. Most importantly it was typically known in advance how big an RoI of a given type was and how much data would be needed for a decision.

With hardware solutions being replaced over time first by RISC workstations and then by generic PCs, L2 trigger algorithms became more and more flexible. In addition the idea of a step-wise and early rejection became more important. It was realized that rather than pushing, i.e. sending, all the

2016 JINST 11 P06008

RoI data blindly to a processing unit, more intelligent software could pull, i.e. request, only the data it needed and stop the processing when it would be clear that the event would not be accepted. This would reduce the overall data transfer bandwidth on the L2 network and allow the use of commercial off-the-shelf network technologies rather than custom built or expensive industrial solutions.

In this pull architecture, the architecture of the current L2 system, the L2 supervisor assigns events to nodes of a farm of processors or to applications running on the nodes (the L2PU applications in the current system) and forwards only the initial RoI pointers from the L1 trigger. The L2 software running in the farm is then free to request any data it wants from the ROS. Since the L2 software sends decisions to the supervisor all the error handling basically comes down to handling time-outs, no matter if there were network errors, an application crashed or any other conceivable problem occurred.

Leaving the decision on what detector data are needed to the L2 algorithm opened up other possibilities, like the full readout of a sub-detector, as it was done e.g. during the initial commissioning phase of the ATLAS detector

4.5.2 Push vs. pull in the Event Builder

The Event Filter algorithms were always assumed to be implemented in software and to be run on the fully built events. However, discussions still occurred related to the actual event building process itself. Early versions of the DataCollection software in 2002 actually had both a push and pull mode foreseen. In the former the DFM would indicate to the ROS PCs to which destination node they should send the event data. In the pull scenario the event building node would send explicit requests to the ROS itself. The latter allowed a straightforward implementation of traffic shaping, which proved to be crucial to avoid overloading the single GbE links into the event building nodes. In the push based system there would have been no single point of control and it would have had to rely on some algorithm that made sure that the completely independent ROS PCs would randomize the sending of their data packets in time. In addition, the pull approach allowed the development of the partial event building procedure (2.6) in a natural way, which in turn had an impressive impact on the online calibration and monitoring possibilities. Partial event building in a push model would have required additional functionality for detecting when an event is completely built.

4.5.3 Push vs. pull in the ROS

The decision to operate both L2 and EB in a pull fashion forces the ROS into a push-pull mode: data are sent by the RODs into the readout buffers and subsequently pulled out by the DAQ/HLT system. For proper operation of the ROS, the data pushed through the data path should always arrive before the corresponding requests driven via the trigger path. However, the latencies on the two paths cannot be easily controlled. This becomes particularly clear when considering that the data fragments are transferred from the RODs to the ROS with strict sequential ordering, while the asynchronous, parallel processing in the L2 farm followed by the parallel event building procedure introduces a reordering of events in terms of latency and sequence.

To overcome the latency uncertainties, dedicated request re-queuing mechanisms have been implemented in the ROS (2.3.4). These allow a request to be internally repeated until either the corresponding data are available or a configurable timeout expires. In general, data requested are available upon receipt of a request because the trigger path uses a distributed, networked architecture, whilst the data path is based on dedicated custom electronics. Latency inversion

conditions, in which data arrive later than the corresponding requests, have only been experienced in very special data-taking test configurations.

4.6 Networking aspects

The choice of Ethernet technology was justified not only by the good price-performance ratio (most Ethernet products are commodity), but also by the fact that long-term multi-vendor support was foreseen for it at the time this decision was taken [198].

The network infrastructure could be built using switching/routing equipment that has become "standard" for data centres: chassis-based devices (built-in redundancy and over 300 Gbit/s switching bandwidth) and "pizza box" devices (1U or 2U format, over 40 Gbit/s switching bandwidth), equipped with GbE and 10GbE ports. A rolling replacement program (with newer generation devices) is foreseen to limit the equipment production lifetime to approximately five years.

Because of the real-time requirements of the DAQ/HLT system the data networks had to be implemented with devices providing high throughput with minimal loss and latency. Devices were selected on the basis of a thorough testing program. A custom GbE tester [199] was built and used to characterize the performance of network devices from the major manufacturers when exposed to DAQ/HLT specific traffic patterns.

Most of the protocols and technologies deployed in the DAQ/HLT networks (2.10.1) are targeted at improving resilience:

- The OSPF and VRRP protocols [103, 104] are widely used to provide redundant connectivity to rack level switches in routed networks. The use of these protocols in the rest of the CERN campus networks favored their adoption in the DAQ/HLT networks too. The current implementation provides redundancy in a master-backup configuration, i.e. no load balancing at all.

- The Multiple Spanning Trees (MST) protocol [105] is the standard OSI layer 2 protocol for providing redundant paths in Ethernet networks that use VLANs [105]. It has been deployed in the data networks in a configuration that implements both redundancy and load balancing.

- Link Aggregation Groups (LAGs [102]) are used to provide redundant point-to-point links (low OSI layer 2) between switches and even between switches and PCs. Depending on the requirements of the nodes interconnected by a LAG different methods of load balancing have been used, ranging from "active backup" [81] to per-frame load balancing. The optimization of the usage of all the links inside a LAG proved to be tedious and for some load balancing algorithms the order of frames is no longer conserved. Thus, if the main requirement is high bandwidth, and redundancy is of second order it is preferable to use a higher speed link, if possible.

Though initially disabled, Quality of Service (QoS) was deployed in the BackEnd network (high priority for SFO↔EFD traffic) to eliminate undesired interference between SFI↔EFD and SFO↔EFD traffic.

Monitoring of the DAQ/HLT networks started by using CA Spectrum [108] and has gradually evolved to use a set of tools (2.10) that provide more detailed statistics required for troubleshooting: the per port traffic statistics are gathered and stored and are complemented by per flow statistics. All information is easily accessible through the NET-IS web interface [110].

2016 JINST 11 P06008

4.7 DAQ/HLT software

4.7.1 History

The roots of the current TDAQ software go back to the mid 1990s. Following the work in various R&D projects [191] leading up to the ATLAS Technical Proposal in 1994 [200], there was a first effort that culminated in the so-called Prototype '-1' software [201]. This software was focused mainly on issues related to run control and configuration as well as the readout part of the DAQ system (i.e. the interface to the detectors) and the event building. Many of the software components developed at this time are still in use today, often modified or updated.

Tests related to the HLT were mostly concerned with the performance of network interconnects. They often used stand-alone code before the so-called Reference Software provided a common framework in 1998 that was subsequently adopted by all the various proposals for the ATLAS networking technology. Starting from 2001 an effort to combine the Reference Software with the Prototype '-1' software resulted in the development of the so-called Data Collection software. This framework provided all the data flow applications for HLT and event building that are still in use today. By 2004 there was a common TDAQ software release available, which was used for the first time in the 2004 test beam at CERN.

Before the switch of the ATLAS offline software to C++ all physics related code was available in Fortran only, with a few notable exceptions written from the beginning in C in view of online use. The use of the Gaudi framework [31] with its strict separation of interfaces and implementation allowed it to be re-used for both the L2 and EF applications. This in turn allowed development of trigger algorithms to be mostly decoupled from the online environment as far as debugging or performance studies are concerned. Trigger algorithm developers are in fact working almost exclusively in the normal offline environment which allows a much broader participation of ATLAS collaborators.

4.7.2 Software development process

From the beginning the online software was using C and C++ rather than Fortran. There was also an emphasis on using best practices for software design that resulted for a while in the use of commercial design tools. In most cases the use of expensive specialized tools has given way to a much more practical application of the principles behind them. In the following we discuss various aspects of the software development process and how it has changed over the years.

CVS [119] was used almost exclusively for concurrent software version control from the beginning, as it was the only widely available free tool for this purpose. While there was a surge of new tools in this area in the last ten years, ATLAS stayed with CVS until 2008 and then switched to Apache™ Subversion® (SVN) [202], mainly because of scalability problems of CVS. SVN was seen as the natural successor and also the least disruptive for users, because of its similarity to CVS. After extensive tests the actual conversion happened for all of ATLAS within the same week, preserving all history in the case of the TDAQ software.

Over the lifetime of the TDAQ software two major build systems were used. The first was an ATLAS specific version of the Software Release Tools (SRT) that originated at the BaBar experiment [203]. This was replaced ATLAS wide in 2001 with a new tool, Configuration Management Tool (CMT [204]), which was seen as more user friendly and with support available from within the collaboration.

2016 JINST 11 P06008

4.7.3 Operating systems and compilers

During the prototype phase, a number of hardware and software platforms for TDAQ applications were studied. This started with commercial UNIX implementations like SunOS/Solaris for Sparc-powered workstations [205] and real-time LynxOS [206] running on PowerPC read-out single-board computers. Later, with the growing popularity of Intel-based PCs (i386 architecture) with Windows or Linux as operating system, the main effort was put into validation of these platforms for both front-end and back-end TDAQ applications. The main platform has become 64-bit Linux SLC5 (a flavor of RedHat Enterprise Linux 5 with Linux kernel 2.6) [59] running on modern generation PCs with Intel XEON multi-core processors and 32-bit Linux running on the ROS PCs and on Single Board Computers in VME crates (RoI Builder, ROD crates). As the standard compiler on Linux platforms the GNU C/C++ compiler suite (version 4.3) was chosen. To maintain code compatibility and to follow the evolution of the C++ standard, the Intel CC compiler is also used for development. This compiler is especially interesting in view of fully exploiting the parallelism and vectorization capabilities of modern Intel processors.

4.7.4 Controls and configuration

CORBA, proposed in 1991 by the Object Management Group (OMG) [32] as an open standard for distributed object computing, is used as the software communication layer for control and configuration of the distributed TDAQ system. It standardizes many common network programming tasks such as object registration, location, activation, parameter marshalling and de-marshalling, and operation dispatching. CORBA supports most widely used programming languages, for example C, C++ and Java; as well as a number of scripting languages, for example Python. Different implementations of the CORBA standard are able to communicate with each other. This interoperability is one of the most important advantages of CORBA and allows for example different CORBA brokers to be used for C++ and Java applications. CORBA provides a high-level object-oriented paradigm for distributed programming and hides the real complexity behind the CORBA API. This substantially simplifies the development and maintenance of distributed code and reduces the testing time. To choose a suitable CORBA broker implementation for the ATLAS TDAQ project, several brokers have been evaluated [207], resulting in the decision to use omniORB [33] as broker for C++ applications and JacORB [34] as broker for Java applications.

4.7.5 Monitoring and error/status reporting

Based on the choice to use CORBA as inter-process communication technology (4.7.4), the basic infrastructure containing custom control, configuration and monitoring packages has been developed and implemented on top of CORBA. Evaluation of existing messaging systems such as the ALEPH Message System [208], the EPICS Alarm Handler [209] or the OPAL message reporting system [210] and of existing error reporting systems such as EMU (Error Message Utility) [211] helped in understanding, defining and adjusting the requirements for a message and error reporting system which would properly serve our needs [212]. A first implementation of the Message Reporting System (MRS) was available since the early stages of the project [213]. Since then no major problems have been encountered using this implementation. However, updates are often necessary due to the overall project evolution.

2016 JINST 11 P06008

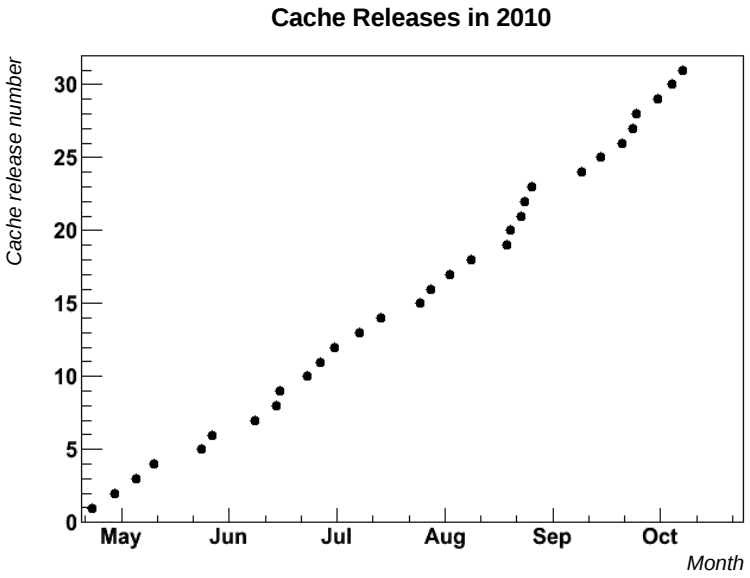


Figure 89. In 2010 frequent changes of the HLT cache release were necessary to adapt to the rapidly increasing luminosity of LHC and the changing beam conditions in the period from May to November. Despite the stability requirements for the underlying base releases, all necessary changes for HLT could be absorbed in these cache releases and rapidly deployed in the experiment.

4.7.6 Offline software in an online environment

The use of offline software in the online environment implies a tight coupling of the two software domains, mainly caused by base libraries, software infrastructure environments and core software packages that are used in both domains. Examples are software components from the LCG releases [100] such as ROOT [133], the common raw data format library from the online group and the basic Athena core software components from the offline framework (2.9). A careful synchronization of the versions of common software components is therefore required. The dependencies go even further and imply also the use of the same version of the operating system in the trigger farms and in the farms used for offline event reconstruction. In ATLAS these questions and the global release schedules are regularly discussed in a "Global Release Coordination" forum, with members from the online, offline and detector communities. There the different release plans are synchronized and the changes for common software versions are scheduled.

In the online domain changes of the major base release versions very rarely happen during the running period. This is imposed by the requirement for operational stability and by the dependencies of the detector software on the basic online control and data flow software. Since also parts of the HLT framework software layer depend on these components and since it takes an extended verification process before a new major HLT base release is deployed for online running, the offline release version, which is associated with an HLT base release, also changes very infrequently in the online domain. Bug fixes and new functionalities for HLT software components are therefore

absorbed in a software cache on top of the base HLT release. These "cache release" versions change much more frequently (see figure 89). This implies however that new features from the offline domain need to be eventually back ported to the active HLT release version, which implies additional maintenance effort.

Many important points of software development such as code robustness, small memory foot print of programs, performance optimization and adaption to changing hardware environments are common to the online and offline software domains. The sharing of experience and the development of common tools and testing platforms is therefore mutually beneficial for both domains and avoids the doubling of effort for similar problems in different software domains. Many results and optimizations obtained in one domain have a direct impact on the other domain owing to the overlap in the code base. Given the complexity of the ATLAS experiment, a significant amount of maintenance effort is also saved by sharing the detector geometry description and the detector data decoding modules between the two software domains. This avoids lengthy synchronization and verification procedures, which would be necessary if offline and online specific implementations were used.

4.7.7 Multi-core processors and multi-threading

The original ATLAS design for the HLT event selection applications exploited event parallelism for obtaining the required event throughput. In L2 concurrent worker threads would execute the selection code in parallel on a subset of the event data, while concurrent processing tasks would work in parallel on full events provided to the EF, with each worker thread and processing task dealing with one event. Parallelism is thus achieved by processing multiple events at the same time and not by parallelizing the reconstruction of a single event. When the HLT architecture was designed, a typical HLT processor consisted of a dual CPU machine with a single compute core per CPU. It was foreseen to start on each L2 processor one instance of an L2PU application with up to three worker threads and two instances of EFPUs per EF processor. The expectation was that increasing clock frequencies would result in increased performance of these CPUs. However, this did not occur, the performance has increased by the introduction of new CPU architectures with multiple compute cores per CPU. The end of the "frequency scaling area" and the introduction of multi-core CPUs posed in principle no problem to the basic HLT design, only the number of event selection threads and processes needed to be adapted to the increasing number of cores.

The use of multi-core CPUs seemed to favor especially multi-threaded HLT selection code. However, it quickly became clear that it is very difficult to maintain thread-safe HLT event selection code in a development environment with many contributors from different subsystems and with different experience. The availability of the same algorithm interfaces in HLT allowed a very fast development and deployment of new selection chains by reusing many components already developed in the offline domain. These components were not always designed to run in a multi-threaded environment and a redesign often had far reaching consequences for key core software components of the offline domain. While in the beginning certain selection chains could run multi-threaded, it proved to be very difficult to maintain this capability over release cycles. Another major problem arose from the basic system libraries themselves. During tests it was observed that the event throughput in the L2PU application did not scale in the expected way with the number of worker-threads. This was due to the use of a common memory pool for container objects in the default memory allocation scheme of the Standard Template Library (STL, [214]), which resulted in frequent mutual blocking of the

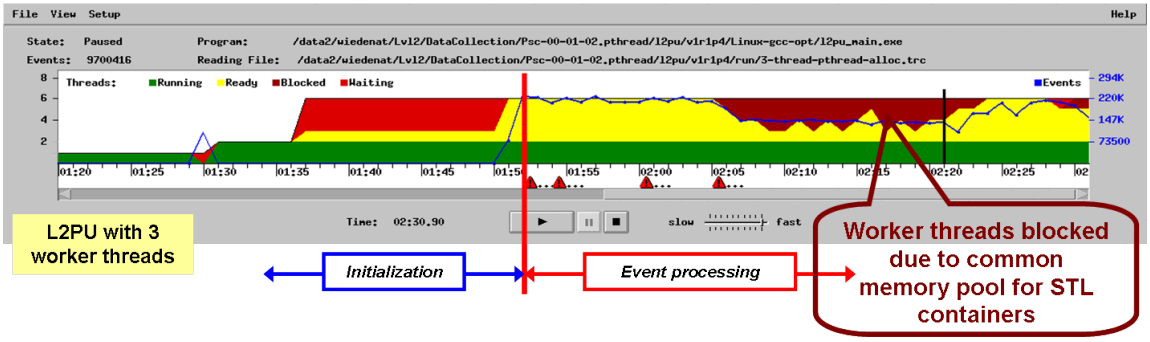


Figure 90. Event processing in three concurrent worker threads. It can be seen that for a substantial fraction of time the worker threads block each other during event processing. This is due to an inefficient memory allocation scheme used in an older version (gcc 2.96) of the *C++ Standard Template Library*.

worker threads as shown in figure 90. The event processing model of L2 favors a scheme where every thread allocates its own memory pool. Such an allocation scheme was available in the STL and after optimizing the code with it, the expected scaling behavior was observed. However, external utility libraries also had to be compiled with this allocation scheme and were not always available in the required form. This made it very difficult to run even thread-safe code in an efficient way. As a consequence the L2PU application is configured to run with one HLT event selection worker thread, but it still uses all the multi-threaded infrastructure for data retrieval, monitoring and run control.

The required throughput is achieved by starting one L2PU or one EFPU instance per CPU core, so that the number of HLT applications per HLT processor is equal to the total number of available CPU cores. An almost linear increase of the event throughput with the number of HLT applications is observed (figure 91), however, the required resources, like available system memory and number of network connections, also increase. Furthermore recent Intel processor models [215] also support hyper-threading (HT), where every CPU core presents to the operating system two virtual cores, which allows in principle a further doubling of the number of selection processes, at the cost of doubled memory needs. However, as figure 91 shows, in a test using Intel E5540 processors [74] only a 20% throughput increase has been achieved.

The main CPU manufacturers implement their recent server processors using a "Non Uniform Memory Architecture" (NUMA), i.e. each CPU has its own local memory and fast links are used to connect the CPUs. While there is little overhead to move processes and threads between the cores on a CPU die, it is typically very costly to move them between different CPUs in such an architecture. The latter can be avoided by either directly pinning the processes to a CPU or by appropriate support from the operating system.

The expected launch of many-core processors in the coming years, i.e. processors providing e.g. 64 or more real hardware threads per processor die, and the possible use of fewer machines with more concentrated computing power to run the HLT selection applications at the same decision rate as now, would also require more bandwidth per node for data input. On the other hand, by running the different selection levels in one node, the input bandwidth requirements could be relaxed because of the increased time spent on an event. The latter scenario is implemented for Run 2 data taking, but will require further enhancements to the framework to optimize the use of multi- and many-core systems (see also refs. [216, 217]).

2016 JINST 11 P06008

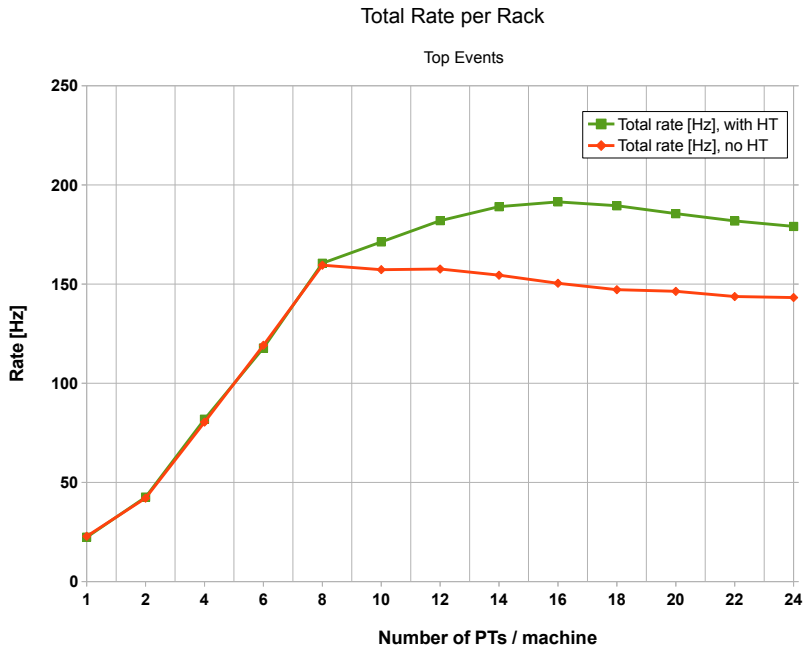


Figure 91. Scaling of the event throughput rate with the number of HLT selection process instances in a rack with 32 dual CPU quad-core machines. Each machine has two Intel Nehalem E5540 processors [74] running at a clock frequency of 2.53 GHz. A Monte Carlo sample of top events was used, which requires a very CPU intensive reconstruction. The test was run with hyper-threading (HT) switched on (upper line) and off (lower line). There is almost linear scaling up to 8 process instances, i.e. one process instance per real hardware thread. Running more selection processes than real hardware threads does not improve the throughput anymore when HT is switched off. Running 16 selection processes, i.e. one process per real and virtual hardware thread, results in an almost 20% increase in event throughput when HT is switched on.

In the early days of ATLAS TDAQ software development the Online, L2 and EF groups worked quite independently of each other and their software had not yet been merged into one overall release. Separate C++ threading packages were developed and used, all based on the POSIX pthreads library (a C language interface, see ref. [218]), at least on Linux platforms. A few attempts at standardizing on one of the packages did not succeed because they were already widely used in their respective domains and the effort of changing was considered to have lower priority than the task of finalizing the TDAQ software. In fact, in the recent past one additional threading package, boost::thread [153], started to be used in some parts of the software. The current expectation is that with the C++11 language standard all software can be converted to use std::threads and associated features.

4.8 System administration

The current structure of the DAQ/HLT system with respect to system management has been derived from the test beam infrastructure implemented and used back in 2004. At that time the system was composed of about 50 computing nodes. As there was no individual functionality defined inside the system, it was decided that their configuration should be identical to ensure that any required task could be performed by any node. Network booting clients seemed to be the most

appropriate solution: it made it easy to propagate changes over the whole cluster, guaranteeing that the cluster remained homogeneous and coherent, and network booting is also quite robust. The boot images were provided to the client nodes by servers, which had the operating system installed on the local disk and were configured manually, without the help of a configuration management tool. Different test labs were put in place during the evolution of the DAQ/HLT computing infrastructure. The preseries testbed (2.16.2) was originally the testbed emulating the final computing environment. Both the system administrators and the DAQ/HLT experts used it to validate the proposed architecture. After the validation, the cluster has been used as a test lab.

4.8.1 History

With the growth of the system it became clear that using a small number of file servers was not a viable option, mostly due to the limitations of NFS. Consequently the number of servers increased and, as the manual installation of a large number of machines can quickly become cumbersome and time consuming, the CERN-wide used configuration management system based on Quattor [145] was adopted. Later in 2009 the management system was migrated to a private SVN-based Quattor service in the ATCN, providing easier management and improved reliability. As of 2011, tests were being performed to migrate to Puppet [146], which is a more flexible and versatile solution. The tests were successful and the migration to Puppet has been completed in 2014 [219].

Also for netbooted nodes, the individual node configuration was abandoned in favour of a tree-like structured arrangement which reflected the grouping of different sub-clusters of nodes with different functionality and requirements. This led to the development of the "Boot With Me" (BWM) project [147]. Following the requests of the detector community, it became apparent that even inside the same sub-cluster there was a need for specific configurations (cronjobs, different services and configurations, etc.). The BWM post-boot scripts allowed individual configurations on top of the general configuration, thus offering versatility to the entire system. In 2012 Puppet was also successfully tested as a replacement for the post-boot scripts, the whole BWM system is being phased out in view of the migration to Puppet.

4.8.2 Services

Also the services provided and used inside the ATCN were shaped by early defining choices. An isolated network for the computing nodes was setup already during the test beam stage. There were two reasons for proceeding this way: the first being that the computing nodes were physically located in a remote area with only low speed CERN networking infrastructure and secondly to restrict the access to the computers, as required by the ATLAS policy. Simple gateway hosts, implementing basic access policies, were used to restrict the access to the internal high-speed network. The gateway access was proven to be very reliable and flexible, so it has been kept for the production system, but with round-robin access to an increased number of gateways.

Another important choice was to use a local LDAP-based authentication system which is also synchronized with CERN IT Active Directory for consistency reasons; indeed the authentication is performed using the user credentials (username and password) provided by CERN IT in conjunction with the role-based architecture described in 2.14.

All the tools used for system administration (either third-party like Nagios and in-house like ConfDB, described in 2.14) use MySQL [43] as the supporting relational database. MySQL has

been preferred above other database servers not just because no other one was available in the ATCN at the time, but also because a large variety of third-party tools, internally used for system administration, offer default support for it. Nowadays Oracle is also available inside the ATCN, but a migration is considered not to bring significant advantages.

4.9 Hardware infrastructure

The DAQ racks in USA-15 provide excellent cooling. Appropriate cable management contributes to this and allows maintenance of each PC without the necessity to disconnect cables (2.18.1). If necessary, ROS PCs can be replaced quickly, which is mandatory as each PC constitutes a single point of failure.

The volume, power and cooling envelope of SDX is fairly constrained but operating experience has shown that the current system operates within those limits. The theoretical cooling limit per rack (2.18.2) is based on a specified water flow of 2 m^3/h giving temperature differences of 4.1 °C and 6 °C for racks on level 2 and on level 1 respectively. In practice none of the racks receive much more than 1 m^3/h. This is reflected in an increased temperature difference: for racks with older processors on level 2 a temperature difference of 6.3 °C has been measured, for racks with newer processors on level 1 it was 7.7 °C. After installation of 80 additional machines on level 1 at the end of 2011 it was necessary and possible to raise the cooling water flow for SDX from 87 to 97 m^3/h, to cope with the additional power dissipation, using part of the flow for USA-15. There are still 23 empty racks on level 1, therefore the cooling infrastructure will need to be revised to cope with the build-up to the full system. However, there are some mitigating tendencies in that newer generation machines produce more compute power for slightly less power.

The construction of SDX permits any water leaking on level 2 to drain through the floor and affect equipment installed on level 1. While this cannot easily be prevented it can be detected and a system of water detection on the ground of both levels will be installed and monitored by DCS. Since the overall health of the TDAQ system is so closely coupled to the efficiency of the power and cooling in SDX the environmental statistics, (temperatures, rack power, etc) are monitored by DCS and are also accessed by the Net-IS program [110] (2.10.2) as a pre-cursor to integrating them into an expert system for a more rapid and accurate diagnosis of any system faults.

5 Conclusions and outlook

The DAQ/HLT system of the ATLAS experiment has been proven over a sustained period of successful data taking. It provides support for smooth operation, extensive monitoring, error detection and handling, and has flexible configuration management facilities. It is built mainly from commercially available rack-mounted server PCs running Linux as operating system and interconnected via GbE and 10GbE. The ROBINs and the RoI Builder are the only custom made parts of the system. High Level Triggering is done completely in software to date. However, a hardware track finder, the FTK, is planned to be added to the system[29].

[29]The FTK will receive a copy of the SCT and pixel detector data sent to the ROBs and will for each L1 accept transfer parameters of track segments to the ROS.

2016 JINST 11 P06008

R&D and design of the system started in the context of the DRDC projects [191] in the early nineties and evolved into the design described in the Technical Design Report submitted in 2003 [30]. During and after this period hardware and software technologies evolved radically.

As expected from Moore's law [194] the computing power available for a given cost and footprint grew throughout the period and continues to grow. In the first instance increase of the CPU clock frequency and architectural improvements were the main factors determining the growth of computing power. Currently the growth is driven by the increase in the number of cores per CPU. The latter trend was not anticipated at the time of submission of the TDR, HLT farm nodes where expected to be machines with dual single-core CPUs running at very high clock speeds. This new trend resulted in many more processes to be controlled than originally foreseen, and also may lead to the available memory per processor and the usage of memory to become an obstacle for full exploitation of future processing power. Hyper-threading, potentially offering more computing power, worsens this problem. Over time packaging and cooling technology have advanced. Early work was done with desktop PCs, nowadays rack-mountable server PCs, widely used in data centres and internet hubs, set the standard. Although the power dissipation per core decreases over time the power dissipation per chassis is increasing as the trend is to pack more motherboards and CPUs into a single server PC, therefore available rack space can be used less efficiently if the cooling power per rack is not increased.

The evolution of Ethernet networking technology resulted in it being a natural choice in 2003, after investigations concerning the applicability of alternative technologies. The performance of switches had been seen as a potential source of problems, in particular with respect to queueing and possible packet loss, this worry is no longer justified for modern technology. Also worries concerning performance degradation due to the use of the TCP/IP protocol have been found to be unjustified.

The limited network bandwidth required was a driving factor for the design and implementation of the RoI driven L2 trigger. For the DAQ/HLT system of the CMS experiment [220] the alternative approach of building events at the full L1 accept rate has been followed at the cost of a more complicated and considerably higher bandwidth network infrastructure than that of ATLAS[30]. The CMS and ATLAS DAQ systems also differ in the organization of the data flow, in particular the CMS system is not based on requesting event fragments but on pushing event fragments to their destination. The question can be raised whether full event building at the L1A rate with the ATLAS approach would have been an option and whether it would have brought advantages. However, it would require a different ROS with the same output bandwidth as input bandwidth and a data flow network with much higher throughput. With the Ethernet technology of 2003 this would have been very expensive and would most likely have required an additional layer of more specialized technology before the Ethernet. It would certainly have been attractive for determining global event properties, such as missing E_T, or for a global track search in the inner tracker with the L2 trigger. However, these features were not considered to be crucial for the physics output of the experiment at the time of the design. It should also be taken into account that a global track search at high luminosity at the L1 accept rate is extremely demanding with respect to the computing resources needed and therefore can only be done for a fraction of the events accepted by L1. Stepwise selection of events is therefore necessary. This makes it natural and attractive to read out the data needed for the first

[30]For ATLAS to have used a similar approach to CMS would have required a network bandwidth more than ten times greater than currently deployed, which would be, because of the larger events, even greater than the bandwidth deployed by CMS.

2016 JINST 11 P06008

steps selectively with a RoI driven data flow between L2 and the ROS, this in view of the reduction of required throughput of the data flow network and of the data handling capacity requirements of the ROS PCs. A missing E_T trigger at L2 could still be implemented using the energy sums embedded in the event fragments of the calorimeters. As described in section 2.3.4 these energy sums are extracted on the basis of special requests by the ROS and forwarded to the L2 trigger. The upgrade of the ROS PCs has made it possible to deploy this trigger at a rate of about 10 kHz.

The implementation of the DAQ/HLT system has benefitted from various developments in software technology, in particular: the development and deployment in and outside high-energy physics of the Linux operating system, the transition to object-oriented programming and multi-threaded programming, and the development of web technology as well as database technology. Although running multi-threaded algorithms in the L2 trigger and the Event Filter in principle allow better utilization of the available resources (in particular of the available memory) it has been found that currently the best approach is to embed sequential algorithms as used in offline processing into a multi-threaded environment, which makes it possible to benefit from the effort spent on development, testing and maintenance of the offline software. However, in the future, with an increasing number of cores per CPU, another approach has to be found, for HLT as well as for offline processing, to make full utilization of the available processing resources possible. For all other aspects of the DAQ/HLT system parallelism is exploited to a high degree. Examples are the tree of run controllers, database access via database proxies, distributed summing of histograms, the multi-threaded ReadOutApplication and SFI application. A multi-threaded version of the SFO application is currently under development.

The DAQ/HLT system is in principle a large distributed computer system. Configuration management, control, error detection and handling, monitoring and system administration, including access management are all essential. Good solutions have been found and implemented and continue to evolve on the basis of the experience gained from operating the system. The use of abstract interfaces and of a plugin architecture for applications provide a large degree of flexibility with respect to deployment and re-use of the software. This facilitates straightforward tests of the software in test setups, which can be as small as a single machine, and re-use for example in the implementation of the new HLT architecture where L2 and EF processing and event building no longer are performed by separate farms and which is to be deployed in Run 2 [221, 222].

The luminosity will continue to increase gradually over time. Predictions of RoI request rates on the basis of cost monitoring information have been shown to be quite accurate. With the trigger menus used in 2012 the request rates are at levels that are close to or exceed what the original ROS PCs could handle, but the upgrade of the ROS PCs prevented this becoming a problem. Further upgrades of the system will consist of replacement of HLT farm nodes and of extension of the HLT farms and also of introduction of new ROBINs and associated ROS PCs as well as the introduction of 10GbE networking at the level of the ROS PCs. Finally adoption of the new HLT architecture can be expected to result in an improved utilization of the available processing resources.

Acknowledgments

We thank CERN for the very successful operation of the LHC, as well as the support staff from our institutions without whom ATLAS could not be operated efficiently.

We acknowledge the support of ANPCyT, Argentina; YerPhI, Armenia; ARC, Australia; BMWFW and FWF, Austria; ANAS, Azerbaijan; SSTC, Belarus; CNPq and FAPESP, Brazil; NSERC, NRC and CFI, Canada; CERN; CONICYT, Chile; CAS, MOST and NSFC, China; COLCIENCIAS, Colombia; MSMT CR, MPO CR and VSC CR, Czech Republic; DNRF, DNSRC and Lundbeck Foundation, Denmark; IN2P3-CNRS, CEA-DSM/IRFU, France; GNSF, Georgia; BMBF, HGF, and MPG, Germany; GSRT, Greece; RGC, Hong Kong SAR, China; ISF, I-CORE and Benoziyo Center, Israel; INFN, Italy; MEXT and JSPS, Japan; CNRST, Morocco; FOM and NWO, Netherlands; RCN, Norway; MNiSW and NCN, Poland; FCT, Portugal; MNE/IFA, Romania; MES of Russia and NRC KI, Russian Federation; JINR; MESTD, Serbia; MSSR, Slovakia; ARRS and MIZŠ, Slovenia; DST/NRF, South Africa; MINECO, Spain; SRC and Wallenberg Foundation, Sweden; SERI, SNSF and Cantons of Bern and Geneva, Switzerland; MOST, Taiwan; TAEK, Turkey; STFC, United Kingdom; DOE and NSF, United States of America. In addition, individual groups and members have received support from BCKDF, the Canada Council, CANARIE, CRC, Compute Canada, FQRNT, and the Ontario Innovation Trust, Canada; EPLANET, ERC, FP7, Horizon 2020 and Marie Skłodowska-Curie Actions, European Union; Investissements d'Avenir Labex and Idex, ANR, Region Auvergne and Fondation Partager le Savoir, France; DFG and AvH Foundation, Germany; Herakleitos, Thales and Aristeia programmes co-financed by EU-ESF and the Greek NSRF; BSF, GIF and Minerva, Israel; BRF, Norway; the Royal Society and Leverhulme Trust, United Kingdom.

The crucial computing support from all WLCG partners is acknowledged gratefully, in particular from CERN and the ATLAS Tier-1 facilities at TRIUMF (Canada), NDGF (Denmark, Norway, Sweden), CC-IN2P3 (France), KIT/GridKA (Germany), INFN-CNAF (Italy), NL-T1 (Netherlands), PIC (Spain), ASGC (Taiwan), RAL (U.K.) and BNL (U.S.A.) and in the Tier-2 facilities worldwide.

A Tables

Table 8. Optical connections. Fibers for which no use is specified are spare fibers. The large number of spare fibers between USA-15 and SDX1 is because it was initially foreseen to connect each ROS PC with 2 fibers to switches in SDX1, which would have required 302 fibers.

USA-15: 1932 fiber pairs, length 12.5–33 m	1583 ROD-ROBIN S-link 7 L1-RoIB S-link
USA15 - SDX1: 360 fiber pairs, length 150 m 12 cables of 60 fibers each	5 RoIB - L2SV S-link 36 x 10GbE for DC Network 2 x 10GbE for ATCN
SDX - CERN computer centre: 10 fiber pairs	4 x 10GbE

2016 JINST 11 P06008

Table 9. Central network switches. The Brocade MLXe16 switch was added as second BackEnd Network switch during the 2011 - 2012 winter shutdown.

Model and Type	total	usage
Force10 E600 [223]	2	Control Network
	1	BackEnd Network
Force10 E1200 [223]	2	DataCollection Network
Brocade MLXe 16 [224]	1	BackEnd Network

Table 10. Event Builder and HLT nodes in SDX1, configuration of October 2011. The amount of memory per core is 2 GB, and therefore per node either 16 or 24 GB for quad-core and hex-core CPUs respectively. In the Bandwidth per rack column, DC refers to the DataCollection Network and BE to the BackEnd Network. Each rack of HLT nodes also contains a file server consisting of a dual quad-core node with either 750 GB or 1.5 TB disk space. All CPUs are Intel CPUs [215].

Type of node CPUs per node	Racks	Bandwidth per rack	Nodes/cores Chassis	Rack	Total
SFI 2 x E5620 2.40 GHz	3	32 x GbE DC 32 x GbE BE	1/8	16/64	48/192
XPU 2 x E5420 2.50 GHz	11	1 x 10GbE DC 2 x GbE BE	1/8	32/256	341/2728 XPU 11/88 services
XPU 2 x X5650 2.67 GHz	12	1 x 10GbE DC 1 x 10GbE BE	4/48	32/384	372/4464 XPU 12/144 services
XPU 2 x X5650 2.67 GHz	12	1 x 10GbE DC 1 x 10GbE BE	4/48	40/480	468/5616 XPU 12/144 services
EF 2 x E5540 2.53 GHz	10	1 x 10GbE BE	4/32	32/256	310/2480 EF 10/80 services
EF 2 x X5650 2.67 GHz	4	1 x 10GbE BE	4/48	32/384	124/1488 EF 4/48 services

B Definitions

Application Self-contained program
Chassis Rack-mountable mechanical structure housing switches or motherboards
Core Single processor in a CPU
CPU Integrated circuit with at least one processor core
Event All data corresponding to one L1 accept
Fragment Part of the event data, buffered in a single ROB
Hyper-Threading Support in hardware for fast switching between threads
Motherboard Printed circuit board with sockets for one or more CPUs and for memory
Multi-core CPU CPU with two or more processor cores

Node	Network endpoint running an operating system
Partition	Self-contained slice of the TDAQ system (2.1.2)
Plugin	Dynamically loadable part of an application
ReadoutApplication	Application running on the ROS PCs
Segment	Independently controllable and configurable part of a partition (2.1.2)
TDAQ resource	Part of the TDAQ system that can individually be enabled or disabled (2.11.3)
Thread	Set of instructions to be executed sequentially
Word	32 bits

C Acronyms

ACR	ATLAS Control Room
ADC	Analog to Digital Converter
AFS	Andrew File System
ALFA	Absolute Luminosity For ATLAS (forward scintillating fiber tracker)
AM	Access Management
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
Athena	Offline software framework, based on the Gaudi framework
athenaMT	L2PU emulator using Athena framework
athenaPT	EFPT emulator using Athena framework
ATLAS	A Toroidal LHC ApparatuS
ATCN	ATLAS Technical and Control Network
BC	Bunch-Crossing
BCId	Bunch-Crossing Identifiier
BCM	Beam Conditions Monitor
BCR	Bunch Counter Reset
BOF	Beginning Of Fragment
BWM	Boot With Me: allows individual configurations on top of the general configuration of netbooted machines
CANbus	Controller Area Network bus
CASTOR	CERN Advanced STORage manager
CERN	European Organization for Nuclear Research
CFS	Central File Server
CLIPS	C Language Integrated Production System
CMT	Configuration Management Tool
COOL	ATLAS-wide conditions database, also: CLIPS Object Oriented Language
ConfDB	MySQL DataBase containing hardware and configuration parameters of computer equipment
CORBA	Common Object Request Broker Architecture
CORAL	COmmon Relational Abstraction Layer
COTS	Commodity/Commercial Off-The-Shelf
CP	Cluster Processor (L1 trigger, electron/photon and tau cluster finding algoritms)
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CRD	Control Room Desktop
CSC	Cathode Strip Chamber
CTP	Central Trigger Processor (L1 trigger)
CVS	Concurrent Versions System
DAL	Data Access Library
DAQ	Data AcQuisition system
DB	DataBase
DBMS	DataBase Management System
DC	Data Collection
DCS	Detector Control System
DFM	Data Flow Manager
DHCP	Dynamic Host Configuration Protocol
DMA	Direct Memory Access
DPM	Dual Port Memory

2016 JINST 11 P06008

DQ	Data Quality
DQM	Data Quality Monitoring
DQMC	Data Quality Monitoring Configurator
DQMD	Data Quality Monitoring Display
DQMF	Data Quality Monitoring Framework
DRDC	Detector Research and Development Committee, set up at CERN in July 1990 last meeting in January 1995
DSP	Digital Signal Processor
DSS	Detector Safety System
DVS	Diagnostic and Verification System
EB	Event Builder
ECR	Event Counter Reset
EDM	Event Data Model
EF	Event Filter
EFD	Event Filter Data flow component
EFP	Event Filter Processort
EFPT	Event Filter Processing Task, same as EFPU
EFPU	Event Filter Processing Unit
emon	event monitoring
EOF	End Of Fragment
ERS	Error Reporting Service
FCal	Forward Calorimeter
FCAPS	Fault, Configuration, Accounting, Performance and Security management
FIFO	First-In First-Out memory or strategy
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
GbE	Gigabit Ethernet
GNAM	Event analysis framework
GPN	CERN General Purpose Network
GPS	Global Positioning System
GUI	Graphical User Interface
HEC	Hadronic End-cap Calorimeter
Histmon	package for updating and periodic publishing of histograms
HLT	High Level Trigger
HOLA	High speed Optical Link for ATLAS
ID	Inner Detector
IGUI	Integrated Graphical User Interface
IP	Internet Protocol
IPC	Inter-Process Communication
IPMI	Intelligent Platform Management Interface
IS	Information Service
ITHistSvc	abstract Athena interface defining the histogram service interface
JEP	Jet/Energy Processor (L1 trigger)
JTAG	Joint Task Action Group
KDE	K Desktop Environment
L1A	L1 Accept
L1	First level trigger
L1Id	L1 Identifier
L1R	L1 Result
L2	Second level trigger
L2PU	L2 Processing Unit (application)
L2RH	L2 Result Handler (application)
L2SV	L2 SuperVisor (application)
LAG	Link Aggregation Group
LAN	Local Area Network
LanDB	CERN central networking database
LAr	Liquid Argon
LCG	LHC Computing Grid
LDAP	Lightweight Directory Access Protocol
LFS	Local File Server
LTP	Local Trigger Processor module

2016 JINST 11 P06008

LHC	Large Hadron Collider
LUCID	LUminosity measurement using a Cherenkov Integrating Detector
NIC	Network Interface Controller
MDA	Monitoring Data Archive
MDT	Monitored Drift Tubes
MON	MONitoring node
MRS	Message Reporting System
MST	Multiple Spanning Tree Protocol
MSTi	Multiple Spanning Tree interface
MySQL	RDBMS making use of SQL
MUCTPI	L1 MUon to Central Trigger Processor Interface
Nagios	Nagios Ain't Gonna Insist On Sainthood: system and network monitoring
NFS	Network File System
NSG	Network Service Gateway
NTP	Network Time Protocol
OH	Online Histogramming Service
OHP	Online Histogramming Presenter
OHS	Online Histogramming Service
OKS	Object Kernel Support
OMD	Operational Monitoring Display
OMG	Object Management Group
OS	Operating System
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
PCI	Peripheral Component Interconnect
PCIe	Peripheral Component Interconnect Express
PEB	Partial Event Building
PLC	Programmable Logic Circuit
PHY	Implementation of the PHYsical layer of the OSI model
PMG	Process ManaGer
PP	PreProcessor (L1 trigger)
PSU	Power Supply Unit
PT	Processing Task
PU	Processor Unit
QoS	Quality of Service
RAC	Real Application Cluster
RAID	Redundant Array of Independent Disks
RF2TTC	CTP Interface for receiving LHC clock signals
ROB	ReadOut Buffer
ROBIN	PCI or PCIe card with ReadOut Buffers
ROC	ReadOut Crate
ROD	ReadOut Driver
RoIB	Region of Interest Builder
RoI	Region of Interest
ROL	ReadOut Link
ROS	ReadOut System
pROS	pseudo ROS: alternative name for L2RH
RBAC	Role Based Access Control
RC	Run Control
RCC	ROD Crate Controller
RCD	ROD Crate DAQ
RDB	Remote DataBase server
RDBMS	Relational DataBase Management System
RFC	Request For Comment
ROOT	Object-Oriented Data Analysis Framework
RMON	Remote MONitoring node
RPC	Resistive Plate Chamber
RPM	Red Hat Package Manager
RRD	Round Robin Database
SBC	Single Board Computer
SCR	Secondary Control Room

SCT	SemiConductor Tracker
SCX	Surface control room
SDX1	Surface counting room
SELinux	Security-Enhanced Linux
SFI	Sub-Farm Input: application running on event building node
SFO	Sub-Farm Output: application running on data logging farm node
sFlow	sampling technology fot monitoring network traffic
SLC	Scientific Linux CERN
S-LINK	Simple LINK
SNMP	Simple Network Management Protocol
SMP	Symmetric MultiProcessor
SMS	Short Message Service
SQL	Structured Query Language
SQLite	Structured Query Language
STL	Standard Template Library
STP	Spanning Tree Protocol
SVN	SubVersioN revision control system
TCP	Transmission Communication Protocol
TDAQ	Trigger and Data AcQuisition
TDR	Technical Design Report
TGC	Thin Gap Chamber
TRT	Transition Radiation Tracker
TTC	Timing, Trigger, and Control
TTC2LAN	Application emulating an L2SV
TTCex	Trigger, Timing, and Control laser transmitter
TTCRx	Trigger, Timing, and Control Receiver ASIC
TTCvi	Timing, Trigger, and Control VME interface module
U-Boot	Universal Bootloader
UDP	User Datagram Protocol
UPS	Uninterruptible Power Supply
VLAN	Virtual LAN
VME	Versa Module Eurocard
VRRP	Virtual Router Redundancy Protocol
WMI	Web Monitoring Interface
XON	Message or signal indicating that data transmission can be resumed
XOFF	Message or signal indicating that data transmission has to be halted
US15	Underground service area
USA15	Underground counting room
UX15	Experimental cavern
VLAN	Virtual Local Area Network
XML	Extensible Markup Language
XPU	Processing Unit that can be configured as L2 processor and as EF processor
ZDC	Zero Degree Calorimeter

2016 JINST 11 P06008

References

[1] L. Evans and P. Bryant, *LHC Machine*, 2008 *JINST* **3** S08001.

[2] *LHC performance and statistics*, https://lhc-statistics.web.cern.ch/LHC-Statistics/.

[3] The ATLAS Collaboration, *The ATLAS Experiment at the CERN Large Hadron Collider*, 2008 *JINST* **3** S08003.

[4] S. Ask et al., *The ATLAS central level-1 trigger logic and TTC system*, 2008 *JINST* **3** P08002.

[5] RD12 Project collaboration, B.G. Taylor, *TTC distribution for LHC detectors*, *IEEE Trans. Nucl. Sci.* **45** (1998) 821.

[6] B.G. Taylor, *Timing distribution at the LHC*, in *Proc. of the* 8th *Workshop on Electronics for LHC Experiments*, CERN-LHCC-2002-034, Colmar France (2002), pg. 63 [http://ttc.web.cern.ch/TTC/LECC02.pdf].

[7] *TTC system*, http://ttc.web.cern.ch/TTC/.

[8] *RF2TTC interface module*, http://ttc-upgrade.web.cern.ch/ttc-upgrade/New_system/RF2TTC.htm.

[9] P. Borrego Amaral et al., *The ATLAS Local Trigger Processor (LTP)*, *IEEE Trans. Nucl. Sci.* **52** (2005) 1202.

[10] Ph. Farthouat and P. Gallno, *TTC-VMEbus Interface*, https://edms.cern.ch/document/110746.

[11] B.G. Taylor, *TTC laser transmitter (TTCex, TTCtx, TTCmx) User Manual*, http://ttc.web.cern.ch/TTC/TTCtxManual.pdf.

[12] J. Christiansen et al., *TTCrx Reference Manual*, http://ttc.web.cern.ch/TTC/TTCrx_manual3.11.pdf.

[13] Liquid Argon Back End Electronics collaboration, A. Bazan et al., *ATLAS liquid argon calorimeter back end electronics*, 2007 *JINST* **2** P06002.

[14] Y. Arai et al., *ATLAS Muon Drift Tube Electronics*, 2008 *JINST* **3** P09001.

[15] G. Aad et al., *ATLAS pixel detector electronics and sensors*, 2008 *JINST* **3** P07007.

[16] ATLAS TRT collaboration, E. Abat et al., *The ATLAS TRT electronics*, 2008 *JINST* **3** P06007.

[17] H.C. van der Bij, R.A. Mclaren, O. Boyle and G. Rubin, *S-link, a data link interface specification for the LHC era*, *IEEE Trans. Nucl. Sci.* **44** (1997) 398.

[18] *The S-LINK data-link*, http://hsi.web.cern.ch/HSI/s-link/.

[19] P. Farthouat, *ATLAS electronics: an overview*, *Int. J. Mod. Phys.* **A 25** (2010) 1761.

[20] A. dos Anjos et al., *The raw event format in the ATLAS Trigger & DAQ*, ATL-D-ES-0019, https://edms.cern.ch/document/445840/4.0e.

[21] R. Achenbach et al., *The ATLAS level-1 Calorimeter Trigger*, 2008 *JINST* **3** P03001.

[22] The ATLAS Collaboration, J.T. Childers, *ATLAS level-1 calorimeter trigger hardware: initial timing and energy calibration*, *J. Phys. Conf. Ser.* **293** (2011) 012061.

[23] The ATLAS Collaboration, M. Wessels, *Calibration and Performance of the ATLAS Level-1 Calorimeter Trigger with LHC Collision Data*, *Phys. Proc.* **37** (2012) 1841.

[24] F. Anulli et al., *The Level-1 Trigger Muon Barrel System of the ATLAS experiment at CERN*, 2009 *JINST* **4** P04010.

[25] The ATLAS Collaboration, S.X. Oda, *Commissioning of the ATLAS level-1 endcap muon trigger system*, *Nucl. Instrum. Meth.* **A 623** (2010) 522.

[26] The ATLAS Collaboration, T. Pauly, *The ATLAS Level-1 Central Trigger System in operation*, *J. Phys. Conf. Ser.* **219** (2010) 022017.

[27] The ATLAS Collaboration, M. Stockton, *The ATLAS Level-1 Central Trigger*, 2011 *JINST* **6** C01075.

[28] The ATLAS Collaboration, C. Gabaldon, *Performance of the ATLAS Trigger System*, 2012 *JINST* **7** C01092.

2016 JINST 11 P06008

[29] The ATLAS Collaboration, *Performance of the ATLAS Trigger System in* 2010, *Eur. Phys. J.* **C 72** (2012) 1849 [arXiv:1110.1530].

[30] The ATLAS Collaboration, *ATLAS High-Level Trigger, Data-Acquisition and Controls: Technical Design Report*, CERN-LHCC-2003-022, CERN, Geneva Switzerland (2003) [ATLAS-TDR-016].

[31] *The Gaudi Project*, http://proj-gaudi.web.cern.ch/proj-gaudi/.

[32] *Object Management Group*, http://www.omg.org/.

[33] *omniORB, a CORBA object request broker for C++ and Python*, http://omniorb.sourceforge.net/.

[34] *Java implementation of the OMG's CORBA standard*, http://www.jacorb.org/.

[35] *ATLAS DAQ/HLT Software Large Scale Functionality and Performance Tests July* 2005, https://edms.cern.ch/document/685256, ATL-D-TR-0003, (2005).

[36] A. dos Anjos et al., *Error Handling and Error Reporting in TDAQ Applications*, https://edms.cern.ch/document/459790, ATL-D-EN-0003, (2004).

[37] M. Caprini et al., *The Message Reporting System in the ATLAS DAQ System*, *Astropart. Part. Space Phys. Detect. Med. Phys. Appl.* **4** (2008) 776.

[38] ATLAS TDAQ collaboration, R. Murillo Garcia and G. Lehmann Miotto, *A revised design and implementation of the ATLAS Log Service package*, *J. Phys. Conf. Ser.* **331** (2011) 042037.

[39] F. Viegas, R. Hawkings and G. Dimitrov, *Relational databases for conditions data and event selection in ATLAS*, *J. Phys. Conf. Ser.* **119** (2008) 042032.

[40] *Oracle Real Application Clusters (RAC)*, http://www.oracle.com/technetwork/database/options/clustering/overview.

[41] *CORAL persistency*, https://twiki.cern.ch/twiki/bin/view/Persistency/Coral.

[42] A. Valassi et al., *LCG Persistency Framework (CORAL, COOL, POOL): Status and Outlook*, *J. Phys. Conf. Ser.* **331** (2011) 042043.

[43] *MySQL*, http://dev.mysql.com/.

[44] *SQLite*, http://sqlite.org/.

[45] A. Valassi et al., *CORAL Server and CORAL Server Proxy: Scalable Access to Relational Databases from CORAL Applications*, *J. Phys. Conf. Ser.* **331** (2011) 042025.

[46] A.X. Widmer and P.A. Franaszek, *A DC-Balanced, Partitioned-Block,* 8*B*/10*B Transmission Code*, *IBM J. Res. Devel.* **27** (1983) 440.

[47] R. Cranfield et al., *The ATLAS ROBIN*, 2008 *JINST* **3** T01002.

[48] *Xilinx Virtex-II documentation*, http://www.xilinx.com/support/.

[49] *PPC*440*GP*-3*CC*466*C processor*, IBM.

[50] *PLX PCI* 9656, http://plxtech.com/products/io/pci9656.

[51] *Das U-Boot — the universal boot loader*, http://www.denx.de/wiki/U-Boot/WebHome.

[52] *Supermicro X6DHE-XB motherboard*, http://www.supermicro.nl/products/motherboard/Xeon800/E7520/X6DHE-XB.cfm.

[53] *Intel* 3.4 *GHz Xeon processor, Irwindale*, http://ark.intel.com/products/28018.

2016 JINST 11 P06008

[54] *Supermicro X7SBE board*, http://www.supermicro.nl/products/motherboard/Xeon3000/3210/X7SBE.cfm.

[55] *Intel Q*9650 *processor*, http://ark.intel.com/products/35428.

[56] *Silicom PEG*-4 *PCIe Gigabit Ethernet interface*, http://silicom-usa.com/Networking_Adapters/PEG4-Quad_Port_Copper_Gigabit_Ethernet_PCI_Express_Server_Adapter_Broadcom_based_50.

[57] *Silicom PEG*-2*i PCIe Gigabit Ethernet interface*, http://silicom-usa.com/Networking_Adapters/PEG2i-Dual_Port_Copper_Gigabit_Ethernet_PCI_Express_Server_Adapter_Intel_based_50.

[58] *Intelligent Platform Management Interface (IPMI)*, http://www.intel.com/content/www/us/en/servers/ipmi/ipmi-home.html.

[59] *Scientific Linux CERN*, http://linux.web.cern.ch/linux/scientific.shtml.

[60] M. Joos, *Contiguous memory allocation under Linux*, https://edms.cern.ch/document/336290, ATL-D-ES-0006, (2006).

[61] S. Gameiro et al., *The ROD crate DAQ software framework of the ATLAS data acquisition system*, *IEEE Trans. Nucl. Sci.* **53** (2006) 907.

[62] R. Blair et al., *The ATLAS High Level Trigger Region of Interest Builder*, 2008 *JINST* **3** P04001 [arXiv:0711.3217].

[63] *Altera APEX* 20*k Devices*, http://www.altera.com/products/devices/apex/apx-index.html.

[64] *HOLA S-link cards*, http://hsi.web.cern.ch/HSI/s-link/devices/hola/.

[65] *FILAR, Quad HOLA S-LINK to* 64*-bit/*66 *MHz PCI Interface*, http://hsi.web.cern.ch/HSI/s-link/devices/filar/.

[66] T. Dai et al., *The ATLAS MDT remote calibration centers*, *J. Phys. Conf. Ser.* **219** (2010) 022028.

[67] P. Bagnaia et al., *Calibration model for the MDT chambers of the ATLAS Muon Spectrometer*, ATL-MUON-PUB-2008-004, CERN, Geneva Switzerland (2008).

[68] W. Vandelli et al., *The ATLAS Event Builder*, *IEEE Trans. Nucl. Sci.* **55** (2008) 3556.

[69] H.P. Beck et al., *An algorithm to determine the oldest still valid event identifier in TDAQ*, https://edms.cern.ch/document/478356, ATL-DQ-EN-0018, (2004).

[70] H.P. Beck et al., *EFIO: Protocol Specification*, https://edms.cern.ch/document/391570, ATL-DQ-ES-0040, (2006).

[71] H.P. Beck et al., *Performance of the final Event Builder for the ATLAS Experiment*, *IEEE Trans. Nucl. Sci.* **55** (2008) 176.

[72] S. Klous et al., *Event streaming in the online system: Real-Time Organization of ATLAS Data*, *IEEE Real Time Conf. Rec.* (2010) 1.

[73] *Intel X*5650 *processor*, http://ark.intel.com/products/47922.

[74] *Intel E*5540 *processor*, http://ark.intel.com/products/37104.

[75] S. Armstrong et al., *Design, deployment and functional tests of the online Event Filter for the ATLAS experiment at LHC*, *IEEE Trans. Nucl. Sci.* **52** (2005) 2846 [*Erratum ibid.* **54** (2007) 758].

2016 JINST 11 P06008

[76] A. Battaglia et al., *The Data-Logging System of the Trigger and Data Acquisition for the ATLAS experiment at CERN*, *IEEE Trans. Nucl. Sci.* **55** (2008) 2607.

[77] *Supermicro X8DTE-F motherboard*, http://www.supermicro.nl/products/motherboard/QPI/5500/X8DT6-F.cfm?IPMI=Y&SAS=N.

[78] *Intel E5520 processor*, http://ark.intel.com/products/40200.

[79] *Adaptec 5805 RAID controller*, http://www.adaptec.com/en-us/support/raid/sas_raid/sas-5805/.

[80] *Intel PRO/1000PT quad port server adapter*, http://support.intel.com/support/network/adapter/1000ptquad/.

[81] The Linux Foundation, *Bonding*, http://www.linuxfoundation.org/collaborate/workgroups/networking/bonding, (2009)

[82] *CASTOR, CERN Advanced STORage manager*, http://castor.web.cern.ch/.

[83] S. George et al., *The ATLAS High Level Trigger Configuration and Steering Software: Experience with 7 TeV Collisions*, PoS(ICHEP 2010)487.

[84] N. Berger et al., *The ATLAS high level trigger steering*, *J. Phys. Conf. Ser.* **119** (2008) 022013.

[85] The ATLAS Collaboration, *ATLAS Computing: Technical Design Report*, CERN-LHCC-2005-022, CERN, Geneva Switzerland (2005) [ATLAS-TDR-017].

[86] PESA Software Group, M. Elsing ed., *Analysis and Conceptual Design of the HLT Selection Software* 013, ATL-DAQ-2002-013, CERN, Geneva Switzerland (2002).

[87] P. Calafiura et al., *The StoreGate: a Data Model for the ATLAS Software Architecture*, in *Proc. Computing in High Energy and Nuclear Physics* 2003, MOJT008, La Jolla CA U.S.A. (2003).

[88] S. Armstrong et al., *Studies for a Common Selection Software Environment in ATLAS: From the Level-2 Trigger to the Offline Reconstruction*, *IEEE Trans. Nucl. Sci.* **51** (2004) 915.

[89] A. dos Anjos et al., *The Configuration System of the ATLAS Trigger*, *IEEE Trans. Nucl. Sci.* **55** (2008) 392.

[90] P.J. Bell et al., *The Configuration System of the ATLAS Trigger*, *IEEE Nucl. Sci. Symp. Conf. Rec.* (2008) 2614.

[91] F. Winklmeier on behalf of the ATLAS TDAQ Collaboration, *Real-time configuration changes of the ATLAS High Level Trigger*, *IEEE Real Time Conf. Rec.* (2010) 1.

[92] W. Fedorko on behalf of the ATLAS Collaboration, *Triggering on 7 TeV collisions with the ATLAS High Level Trigger*, *IEEE Nucl. Sci. Symp. Conf. Rec.* (2010) 1272.

[93] D.W. Miller on behalf of the ATLAS collaboration, *Online Measurement of LHC Beam Parameters with the ATLAS High Level Trigger*, *IEEE Real Time Conf. Rec.* (2010) 1.

[94] R. Bartoldus for the ATLAS Collaboration, *Online Determination of the LHC Luminous Region with the ATLAS High-Level Trigger*, *Phys. Proc.* **37** (2012) 2080.

[95] F. Winklmeier et al., *A system for monitoring and tracking the LHC beam spot within the ATLAS High Level Trigger*, *IEEE Real Time Conf. Rec.* (2012) 1.

[96] P. Werner et al., *ATLAS TDAQ DataFlow L2PU Processing Timeout*, https://edms.cern.ch/document/1095580/1, ATL-DH-ER-0001, (2010).

2016 JINST 11 P06008

[97] P. Pinto, A. dos Anjos and W. Wiedenmann, *AthenaMT and Level-2 Software Integration*, https://edms.cern.ch/document/571749, ATL-DH-EN-0009, (2005).

[98] M. Bosman et al., *AthenaPT (pt_test) and Event Filter Software Integration*, https://edms.cern.ch/document/581296/1, ATL-DH-OR-0002, (2005).

[99] E. Obreshkov et al., *Organization and management of ATLAS software releases*, *Nucl. Instrum. Meth.* **A 584** (2008) 244.

[100] *LCG Applications Area software projects*, http://ep-dep-sft.web.cern.ch/project/packages-releases.

[101] S.N. Stancu et al., *Network Resiliency Implementation in the ATLAS TDAQ System*, *IEEE Real Time Conf. Rec.* (2010) 1.

[102] *Link Aggregation*, IEEE 802.1ax, http://standards.ieee.org/getieee802/802.1.html.

[103] J. Moy, *OSPF Version* 2, RFC 2328, http://www.rfc-archive.org/getrfc.php?rfc=2328, April 1998.

[104] R. Hinden, *Virtual Router Redundancy Protocol (VRRP)*, RFC 3768, http://www.rfc-archive.org/getrfc.php?rfc=3768, April 2004.

[105] *Virtual Bridged Local Area Networks*, IEEE 802.1Q, http://standards.ieee.org/getieee802/802.1.html.

[106] S.M. Batraneanu et al., *Operational Model of the ATLAS TDAQ Network*, *IEEE Trans. Nucl. Sci.* **55** (2008) 687.

[107] B. Martin et al., *Advanced monitoring techniques for a large-scale data-processing network*, *Campus-Wide Informat. Syst.* **25** (2008) 287.

[108] *CA Spectrum® Infrastructure Manager*, http://www.ca.com/us/root-cause-analysis.aspx.

[109] *Apache Thrift™*, http://thrift.apache.org.

[110] D.O. Savu, A. Al-Shabibi, B. Martin, R. Sjon, S.M. Batraneanu and S. Stancu, *Integrated System for Performance Monitoring of the ATLAS TDAQ Network*, *J. Phys. Conf. Ser.* **331** (2011) 052031.

[111] D. Savu et al., *Efficient Network Monitoring for Large Data Acquisition Systems*, in *Proc. of ICALEPCS*2011, http://accelconf.web.cern.ch/AccelConf/icalepcs2011/papers/wepmu036.pdf, Grenoble France (2011), pg. 1153.

[112] T. Oetiker, *RRDtool*, http://www.mrtg.org/rrdtool/.

[113] P. Phaal, P. Panchen and N. McKee, *Inmon Corporation's sFlow: a method for monitoring traffic in switched and routed networks*, RFC 3176, http://www.rfc-archive.org/getrfc.php?rfc=3176, September 2001.

[114] M. Leahu, M. Dobson and G. Avolio, *Access Control Design and Implementations in the ATLAS Experiment*, *IEEE Trans. Nucl. Sci.* **55** (2008) 386.

[115] *OpenLDAP project*, http://www.openldap.org.

[116] G. Avolio, M. Dobson, G. Lehmann Miotto and M. Wiesmann, *The Process Manager in the ATLAS DAQ System*, *IEEE Trans. Nucl. Sci.* **55** (2008) 399.

[117] G. Lehmann Miotto et al., *The ATLAS DAQ System Online Configurations Database Service Challenge*, *J. Phys. Conf. Ser.* **119** (2008) 022004.

2016 JINST 11 P06008

[118] R. Jones, L. Mapelli, Yu. Ryabov and I. Solovev, *The OKS Persistent In-memory Object Manager*, *IEEE Trans. Nucl. Sci.* **45** (1998) 1958.

[119] *CVS, Concurrent Versions System*, http://cvs.nongnu.org.

[120] *CLIPS: a Tool for Building Expert Systems*, http://clipsrules.sourceforge.net/.

[121] D. Liko et al., *Control in the ATLAS TDAQ system*, in *Proc. Computing in High Energy and Nuclear Physics* 2004, ATL-DAQ-2004-013, Interlaken Switzerland (2005), pg. 159 [CERN-2005-002-V2].

[122] A. Kazarov, A. Corso-Radu, G.L. Lehmann Miotto, J.E. Sloper and Yu. Ryabov, *A Rule-Based Verification and Control Framework in ATLAS Trigger-DAQ*, *IEEE Trans. Nucl. Sci.* **54** (2007) 604.

[123] M. Barczyk et al., *Verification and Diagnostics Framework in ATLAS Trigger/DAQ*, in *Proc. Computing in High Energy and Nuclear Physics* 2003, TUGP005, La Jolla CA U.S.A. (2003).

[124] J.E. Sloper, G. Lehmann Miotto and E. Hines, *Dynamic Error Recovery in the ATLAS TDAQ System*, *IEEE Trans. Nucl. Sci.* **55** (2008) 405.

[125] *Swing GUI components*, http://docs.oracle.com/javase/6/docs/api/javax/swing/package-summary.html.

[126] G. Avolio, A. Corso Radu, A. Kazarov, G. Lehmann Miotto and L. Magnoni, *Applications of advanced data analysis and expert system technologies in the ATLAS Trigger-DAQ Controls framework*, *J. Phys. Conf. Ser.* **396** (2012) 012003.

[127] A. Barriuso Poy et al., *The detector control system of the ATLAS experiment*, 2008 *JINST* **3** P05006.

[128] *Nagios*, http://www.nagios.org.

[129] *Esper — Event Stream and Complex Event Processing for Java*, http://esper.codehaus.org/about/esper/esper.html.

[130] G. Lehmann Miotto, L. Magnoni and J.E. Sloper, *The TDAQ Analytics Dashboard: a real-time web application for the ATLAS TDAQ control infrastructure*, *J. Phys. Conf. Ser.* **331** (2011) 022019.

[131] *KDE*, http://www.kde.org.

[132] *KIOSK*, http://techbase.kde.org/KDE_System_Administration/Kiosk/Introduction.

[133] *ROOT project*, http://root.cern.ch/.

[134] The ATLAS Collaboration, P. Renkel, *The Gatherer — a mechanism for integration of monitoring data in ATLAS*, *J. Phys. Conf. Ser.* **219** (2010) 022043.

[135] P. Adragna, A. Dotti, C. Roda, R. Ferrari, W. Vandelli and P.F. Zema, *GNAM: a low-level monitoring program for the ATLAS experiment*, *IEEE Trans. Nucl. Sci.* **53** (2006) 1317.

[136] M. Hauschild, H. Hadavand, R. Kehoe, A. Corso-Radu and S. Kolos, *Data Quality Monitoring Framework for the ATLAS experiment at the LHC*, *IEEE Trans. Nucl. Sci.* **55** (2008) 417.

[137] P.F. Zema, *The Monitoring Data Archiving Service for ATLAS*, *IEEE Nucl. Sci. Symp. Conf. Rec.* (2006) 12.

[138] C. Cuenca Almenar et al., *ATLAS Online Data Quality Monitoring*, *IEEE Real Time Conf. Rec.* (2010) 1.

[139] P. Adragna et al., *GNAM and OHP: Monitoring Tools for ATLAS experiment at LHC*, *IEEE Nucl. Sci. Symp. Conf. Rec.* **1** (2007) 114.

[140] *Qt project*, http://qt-project.org.

2016 JINST 11 P06008

[141] A. Sidoti for the ATLAS TDAQ Collaboration, *Trigger Monitoring at ATLAS*, in *Astroparticle, Particle, Space Physics, Detectors and Medical Physics Applications*, *Proc. of the* 11th *conference* **5** (2010) 516.

[142] *FreeNX technology*, https://sourceforge.net/projects/freenx.berlios/, http://web.archive.org/web/20050803005415/http://freenx.berlios.de:80/.

[143] R. Ospanov on behalf of the ATLAS Collaboration, *Resource utilization by the ATLAS High Level Trigger during* 2010 *and* 2011 *LHC running*, *Phys. Proc.* **37** (2012) 1900.

[144] *NetApp*, http://www.netapp.com.

[145] R. Garcia Leiva et al., *Quattor: Tools and Techniques for the Configuration, Installation and Management of Large-Scale Grid Computing Fabrics*, *J. Grid Comput.* **2** (2004) 313.

[146] *Puppet*, http://www.puppetlabs.com.

[147] A. Adeel-Ur Rehman et al., *System administration of ATLAS TDAQ Computing Environment*, *J. Phys. Conf. Ser.* **219** (2010) 022048.

[148] *Scientific Linux*, https://www.scientificlinux.org.

[149] *Role Based Access Control (RBAC) and Role Based Security*, http://csrc.nist.gov/rbac.

[150] D. Burckhart-Chromek et al., *Testing on a Large Scale: running the ATLAS Data Acquisition and High Level Trigger Software on* 700 *PC Nodes*, in *Proc. Computing in High Energy and Nuclear Physics* 2006*, Mumbai India*, S. Banerjee ed., MacMillan, Mumbai India (2006), pg. 60 [ATL-DAQ-CONF-2006-002].

[151] B. Allongue et al., *The electronics system of the ALFA forward detector for luminosity measurements in ATLAS*, 2012 *JINST* **7** C02034.

[152] *LHC Computing Grid*, http://lcgsoft.cern.ch/.

[153] *BOOST libraries*, http://www.boost.org/.

[154] *RedHat Package Manager*, http://rpm.org.

[155] *RedHat Linux*, http://www.redhat.com.

[156] *Savannah tracking system*, http://gna.org/projects/savane/.

[157] *Canalis is a multi vendor industrial modular power distribution system. It is regulated by the standard iec* 60439-2*: low-voltage switchgear and controlgear assemblies — part* 2*: particular requirements for busbar trunking systems (busways)*, http://www.iec.ch/.

[158] *Twido Programmable Logic Controllers*, http://www.schneider-electric.com/products/ww/en/3900-pac-plc-other-controllers/3920-controllers-plc-for-commercial-machines/533-programmable-controller-twido.

[159] H.P. Beck et al., *ATLAS DAQ/HLT Infrastructure*, in 11th *Workshop on Electronics for LHC and Future Experiments*, Heidelberg Germany (2005), pg. 327 [CERN-2005-011].

[160] *SG*160 *Inrush Current Limiting Thermistors, 'Surge Gard' ™series*, http://www.rhopointcomponents.com/components/circuit-protection/surge-gard-inrush-current-limiters-sg-series.html.

[161] Yu. Ermolin, H. Burckhart, D. Francis and F.J. Wickens, *ATLAS DAQ/HLT rack DCS*, *Nucl. Instrum. Meth.* **A 572** (2007) 59.

2016 JINST 11 P06008

[162] *PLX PEX* 8311, http://www.plxtech.com/products/pci_express/PEX8311/default.asp.

[163] *Intel Threading Building Blocks*, http://threadingbuildingblocks.org/.

[164] The ATLAS Collaboration, *The ATLAS Inner Detector commissioning and calibration*, *Eur. Phys. J.* **C 70** (2010) 787 [arXiv:1004.5293].

[165] The ATLAS Collaboration, *Readiness of the ATLAS Liquid Argon Calorimeter for LHC collisions*, *Eur. Phys. J.* **C 70** (2010) 723 [arXiv:0912.2642].

[166] The ATLAS Collaboration, *Readiness of the ATLAS Tile Calorimeter for LHC collisions*, *Eur. Phys. J.* **C 70** (2010) 1193 [arXiv:1007.5423].

[167] The ATLAS Collaboration, *Commissioning of the ATLAS Muon Spectrometer with Cosmic Rays*, *Eur. Phys. J.* **C 70** (2010) 875 [arXiv:1006.4384].

[168] J. Bystricki et al., *A Model for Sequential Processing in the ATLAS LVL2/LVL3 Trigger*, ATL-DAQ-96-055, CERN, Geneva Switzerland (1996).

[169] S. George et al., *Input Parameters for Modelling the ATLAS Second Level Trigger*, ATL-DAQ-97-070, CERN, Geneva Switzerland (1997).

[170] M. Dobson et al., *Paper Models of the ATLAS Second Level Trigger*, ATL-DAQ-98-113, CERN, Geneva Switzerland (1998).

[171] A. Amadon et al., *Architecture C Performance from Paper Models*, ATL-DAQ-98-106, CERN, Geneva Switzerland (1998).

[172] J. Bystricky and J.C. Vermeulen, *Paper modelling of the ATLAS LVL2 trigger system*, ATL-DAQ-2000-030, CERN, Geneva Switzerland (2000).

[173] E.C. Russell, *SIMSCRIPT II.5 and MODSIM II: a brief introduction*, *Simulation Conference, Proc.* (1991) 62.

[174] A. Bogaerts et al., *Modelling of the ATLAS Data Acquisition and Trigger System*, ATL-DAQ-94-018, CERN, Geneva Switzerland (1994).

[175] S. Hunt et al., *SIMDAQ: A System for Modeling DAQ/Trigger Systems*, *IEEE Trans. Nucl. Sci.* **43** (1996) 69.

[176] C. Hortnagl and S. Hunt, *SIMDAQ Users Guide*, ATL-DAQ-95-041, CERN, Geneva Switzerland (1995).

[177] *The Ptolemy Project, Ptolemy Classic*, http://ptolemy.berkeley.edu/ptolemyclassic/, (1998).

[178] M. Dobson et al., *Ptolemy simulation of the ATLAS level-2 trigger*, ATL-DAQ-2000-039, CERN, Geneva Switzerland (2000).

[179] K. Korcyl et al., *Modeling Ethernet networks for the ATLAS Level-2 trigger*, ATL-DAQ-2000-044, CERN, Geneva Switzerland (2000).

[180] J.C. Vermeulen, *Simulation of Data-Acquisition and Trigger Systems in C++*, in *New Computing Techniques in Physics Research III*, World Scientific Singapore (1994), pg. 107 and EAST note 93-22.

[181] J.C. Vermeulen et al., *Discrete event simulation of the ATLAS second level trigger*, *IEEE Trans. Nucl. Sci.* **45** (1998) 1989.

[182] J.C. Vermeulen, *Computer modelling of the ATLAS LVL2 trigger system*, ATL-DAQ-2000-035, CERN, Geneva Switzerland (2000).

2016 JINST 11 P06008

[183] R. Cranfield et al., *Computer Modeling the ATLAS Trigger/DAQ System Performance*, *IEEE Trans. Nucl. Sci.* **51** (2004) 532.

[184] B.J. Green, J.A. Strong, R. Cranfield and G. Crone, *A second level data buffer with LHC performance*, *Nucl. Instrum. Meth.* **A 360** (1995) 359.

[185] R. Cranfield et al., *Prototyping hardware for the ATLAS readout buffers*, in *Proc.* 4th *Workshop on Electronics for LHC Experiments*, http://www.nikhef.nl/pub/experiments/atlas/daq/Proto-ROBs-LEB98.pdf, Rome Italy (1998), pg. 397 [CERN-LHCC-98-36].

[186] *Summary of prototype RobIns*, https://edms.cern.ch/document/382933, ATL-DQ-ER-0001, (2003).

[187] A. Kugel et al., *A RobIn Prototype for a PCI-Bus based ATLAS Readout-System*, in *Proc.* 9th *Workshop on Electronics for LHC Experiments*, Amsterdam The Netherlands, http://cdsweb.cern.ch/record/703766, CERN 2003-006, (2003), pg. 152.

[188] B. Green, G. Kieft, A. Kugel, M. Muller and M. Yu, *ATLAS trigger/DAQ RobIn prototype*, *IEEE Trans. Nucl. Sci.* **51** (2004) 465.

[189] *ATLAS Insertable B-Layer Technical Design Report*, CERN-LHCC-2010-013, CERN, Geneva Switzerland (2010) [ATLAS-TDR-019].

[190] *Fast TracKer (FTK) Technical Design Report*, CERN-LHCC-2013-007, CERN, Geneva Switzerland (2013) [ATLAS-TDR-021].

[191] *Detector Research & Development Committee (DRDC) Public Documents*, http://committees.web.cern.ch/Committees/obsolete/DRDC/Projects.html.

[192] J. Strong, *Local processing for a farm-based second level trigger at LHC*, in *Proc. Computing in High-energy Physics, San Francisco U.S.A. (*1994*)*, S.C. Loken ed., (1994), pg. 471 [ATL-DAQ-94-021] [LBL-35822] [CONF-940492].

[193] D. Calvet, *A Review of Technologies for the Transport of Digital Data in Recent Physics Experiments*, *IEEE Trans. Nucl. Sci.* **53** (2006) 789.

[194] *Moore's law*, http://www.intel.com/pressroom/kits/events/moores%5Flaw%5F40th.

[195] The ATLAS Collaboration, *ATLAS DAQ, EF, LVL2 and DCS Technical Progress Report and Workplan*, http://atlas.web.cern.ch/Atlas/GROUPS/DAQTRIG/TPR/tpr.html, CERN/LHCC 98-16, CERN, Geneva Switzerland (1998).

[196] IEEE Computer Society, *IEEE Standard for Scalable Coherent Interface (SCI)*, *IEEE Std.* 1596–1992 (1993).

[197] The ATLAS Collaboration, *ATLAS High-Level Triggers, DAQ and DCS Technical Proposal*, http://atlas.web.cern.ch/Atlas/GROUPS/DAQTRIG/SG/TP/tp_doc.html, CERN/LHCC/2000-17, CERN, Geneva Switzerland (2000).

[198] S. Stancu et al., *The use of Ethernet in the Dataflow of the ATLAS Trigger & DAQ*, in *Proc. Computing in High Energy and Nuclear Physics* 2003, MOGT010, La Jolla CA U.S.A. 2003.

[199] M. Ciobotaru, S. Stancu, M.J. LeVine and B. Martin, *GETB — a Gigabit Ethernet Application Platform: its use in the ATLAS TDAQ network*, *IEEE Trans. Nucl. Sci.* **53** (2006) 817.

[200] The ATLAS Collaboration, *ATLAS Technical Proposal*, http://atlas.web.cern.ch/Atlas/TP/tp.html, CERN/LHCC/94-43, CERN, Geneva Switzerland December 15 1994 [LHCC/P2].

2016 JINST 11 P06008

[201] L. Mapelli et al., *The ATLAS DAQ and Event Filter prototype '−1' project*, *Comput. Phys. Commun.* **110** (1998) 95.

[202] *Apache™ Subversion®*, http://subversion.apache.org/.

[203] G. Cosmo, *The BaBar Software Architecture and Infrastructure*, *Nucl. Phys. (Proc. Suppl.)* **B 78** (1999) 732.

[204] *CMT, Code Management Tool*, http://www.cmtsite.net.

[205] *SPARC, Scalable Processor Architecture*, http://www.sparc.org.

[206] *LynxOS real-time operating system*, http://www.lynuxworks.com/rtos/.

[207] S. Kolos, *Evaluation of CORBA implementations*, https://edms.cern.ch/document/403799, ATL-DQ-TN-0018, CERN, Geneva Switzerland (2003).

[208] *Experience with the ALEPH Online System*, CERN-ALEPH-PUB-2001-003, CERN, Geneva Switzerland (2001).

[209] *Experimental Physics and Industrial Control System (EPICS)*, http://www.aps.anl.gov/epics/index.php.

[210] OPAL collaboration, J.T.M. Baines et al., *The data acquisition system of the OPAL detector at LEP*, *Nucl. Instrum. Meth.* **A 325** (1993) 271.

[211] P. Burkimsher, *EMU, the MODEL Error Message Utility*, https://emu.web.cern.ch/emu/, (1990).

[212] *ATLAS DAQ Back-end software User Requirements Document*, ATL-DAQ-98-090, CERN, Geneva Switzerland (1998).

[213] A. Amorim et al., *Use of CORBA in the ATLAS prototype DAQ*, *IEEE Trans. Nucl. Sci.* **45** (1998) 1978.

[214] *STL, Standard Template Library*, http://en.cppreference.com/w/cpp/container.

[215] *Intel processors*, http://ark.intel.com.

[216] S. Binet, P. Calafiura, S. Snyder, W. Wiedenmann and F. Winklmeier, *Harnessing multicores: strategies and implementations in ATLAS*, *J. Phys. Conf. Ser.* **219** (2010) 042002.

[217] S. Jarp, A. Lazzaro and A. Nowak, *The future of commodity computing and many-core versus the interests of HEP software*, *J. Phys. Conf. Ser.* **396** (2012) 052058.

[218] IEEE Computer Society and The Open Group, *Standard for Information Technology — Portable Operating System Interface (POSIX®) Base Specifications, Issue* 7, *IEEE Std.* 1003.1 (2013).

[219] S. Ballestrero et al., *Upgrade and integration of the configuration and monitoring tools for the ATLAS Online farm*, *J. Phys. Conf. Ser.* **396** (2012) 042005.

[220] CMS Collaboration, *The CMS experiment at the CERN LHC*, 2008 *JINST* **3** S08004.

[221] A. Negri on behalf of the ATLAS TDAQ collaboration, *Evolution of the Trigger and Data Acquisition System for the ATLAS experiment*, *J. Phys. Conf. Ser.* **396** (2012) 012033.

[222] N. Garelli on behalf of the ATLAS Collaboration, *The evolution of the Trigger and Data Acquisition System in the ATLAS experiment*, *J. Phys. Conf. Ser.* **513** (2014) 012007.

[223] *Force*10 *network switches*, http://www.force10networks.com.

[224] *Brocade network switches*, http://www.brocade.com.

2016 JINST 11 P06008

The ATLAS TDAQ Collaboration

Abolins M.,[55] Abreu R.,[18] Achenbach R.,[33a] Aharrouche M.,[49] Aielli G.,[78a,78b] Al-Shabibi A.,[18] Aleksandrov I.,[37] Alexandrov E.,[37] Allbrooke B.M.,[10] Aloisio A.,[62a,62b] Alonso F.,[42] Alvarez-Gonzalez B.,[55] Alviggi M.G.,[62a,62b] Amorim A.,[44a,44b] Amram N.,[89] Anders G.,[33] Andreani A.,[56a,56b] Andreazza A.,[56a,56b] Andrei V.,[33a] Anduaga X.,[42] Angelaszek D.,[1] Anjos N.,[44a] Annovi A.,[27a,27b] Antonelli S.,[12a,12b] Anulli F.,[77a] Apolle R.,[67] Aracena I.,[80] Artoni G.,[77a,77b] Ask S.,[19] Åsman B.,[85] Soares Augusto M.,[44b] Avolio G.,[95] Backes M.,[28] Badescu E.,[16a] Baines J.,[74] Ballestrero S.,[38b] Banerjee Sw.,[98] Bansil H.S.,[10] Barnett B.M.,[74] Bartoldus R.,[80] Bartsch V.,[86] Batraneanu S.,[95] Battaglia A.,[9] Bauss B.,[49] Beauchemin P.,[67] Beck H.P.,[9,a] Bee C.,[51] Beemster L.,[63] Begel M.,[15] Belanger-Champagne C.,[52] Bell P.,[28] Bell W.H.,[28] Bellagamba L.,[12a,12b] Bellomo M.,[68a] Ben Ami S.,[88] Bendtz K.,[85a,85b] Benhammou Y.,[89] Beretta M.,[27] Berge D.,[18] Bergeaas-Kuutmann E.,[24] Bernard C.[13] Bernat P.,[47] Bernius C.,[48] Bevacqua V.,[71] Bianchi R.M,[18] Bianco M.,[43a,43b] Biglietti M.,[79a] Bindi M.,[12a,12b] Black K.[13] Blair R.E.,[3] Blumenschein U.,[30] Bock R.,[18,e] Bogaerts A.,[18] Bohm C.,[85a] Boisvert V.,[46] Bold T.,[95a] Bondioli M.,[95] Borer C.,[9] Boscherini D.,[12a] Bosman M.,[6] Bossini E.,[71] Boveia A.,[20] Bracinik J.,[10] Brandt A.G.,[5] Brawn I.P.,[74] Brenner R.,[97] Bressler S.,[88] Brock R.,[55] Brooks W.K.,[81b] Brown G.,[50] Brunet S.,[24] Bruni A.,[12a] Bruni G.,[12a,12b] Bucci F.,[28] Buda S.,[16a] Burckhart-Chromek D.,[18] Buscher V.,[49] Butler J.[13] Buttinger W.,[19] Calvet S.,[49] Campanelli M.,[47] Canale V.,[62a,62b] Canelli F.,[20] Cao T.,[25] Capasso L.,[62a,62b] Caprini M.,[16a] Caramarcu C.,[16] Cardarelli R.,[79a] Carlino G.,[62a] Casadei D.,[59] Casado M.P.,[6] Caughron S.,[55] Cerri A.,[18] Cerrito L.,[45] Chapleau B.,[52] Chavez Barajas C.,[18] Cheatham S.,[52] Childers J.T.,[33] Chiodini G.,[43a] Chislett R.,[47] Chitan A.,[16a] Christidi I.,[47] Ciapetti G.,[77a,77b] Ciodaro Xavier T.,[76a] Citraro S.,[71a,71b] Citterio M.,[56a] Coccaro A.,[29a,29b] Cogan J.,[80] Conde Muiño P.,[44a,44b] Conidi C.,[6] Conventi F.,[62a,b] Cooper B.D,[47] Corradi M.,[12a] Corriveau F.,[52] Corso-Radu A.,[95] Coura Torres R.,[76a] Cranmer K.,[59] Cranfield B.,[47] Crescioli F.,[71] Crone G.,[47] Cuenca Almenar C.,[101] Cummings J.T.,[101] Czyczula Z.,[101] Dam M.,[21] Damazio D.,[15] Dao V.,[28] Darlea G.L.,[72c,18] Davignon O.,[41] Davis A.O.,[74] Davygora Y.,[33] Dawson J.,[3,e] De Cecco S.,[41] De Pedis D.,[77a] De Santo A.,[86] de Seixas J.M.,[76a] Degenhardt J.,[69] Delgado A.,[44a,44b] Dell'Asta L.[13] Dell'Orso M.,[71a,71b] Della Pietra M.,[62a,b] Della Volpe D.,[62a,62b] Demers S.,[101] Demirkoz B.,[6] Deviveiros P.O.,[63] Di Ciaccio A.,[78a,78b] Di Mattia A.,[55] Di Simone A.,[78a,78b] Diaz M.A.,[81a] Dietzsch T.A.,[33] Dinut F.,[16] Dionisi C.,[77a,77b] Dobinson R.,[18,e] Dobson E.,[18] Dobson M.,[18] dos Anjos A.,[100] Dova M.T.,[42] Drake G.,[3] Dufour M.,[52] Dumitru I.,[72c,18] Dunford M.,[18] Ebling A.,[49] Eckweiler S.,[49] Ehrenfeld W.,[24] Eisenhandler E.,[45] Ellis K.V.,[45] Ellis N.,[18] Emeliyanov D.,[74] Enoque Ferreira de Lima D.,[76a] Ermoline Y.,[55] Ernst J.,[1] Etienne F.,[51,e] Etzion E.,[89] Evangelakou D.,[30] Falciano S.,[77a] Farrington S.,[67] Farthouat P.,[18] Faulkner P.J.W.,[10] Fedorko W.,[55] Fellmann D.,[3] Feng E.,[20] Ferrari R.,[68a] Ferrer M.L.,[27] Ferretto Parodi A.,[29a,29b] Fiorini L.,[6] Fischer G.,[24] Fonseca Martin T.,[9] Francis D.,[18] Fratina S.,[69] French S.T.,[19] Front D.,[99] Fukunaga C.,[92] Gabaldon Ruiz C.,[18] Gadomski S.,[28] Garelli N.,[18] Gee C.N.P.,[74] George S.,[46] Giagu S.,[77a,77b] Giannetti P.,[71a,71b] Giorgi M.,[8] Giunta M.,[77a] Giusti P.,[12a,12b] Goebel M.,[24] Gomez Fajardo L.S,[24] Gonçalo R.,[46] Gonzalez Silva L.,[17] Göringer C.,[49] Gorini B.,[18] Gorini E.,[43a,43b] Grabowska-Bold I.,[95] Green B.,[46] Guler H.,[52,57] Haas S.,[18] Haberichter W.,[3] Hadavand H.,[25] Hadley D.R.,[10] Haller J.,[30] Hamilton A.,[28] Hanke P.,[33a] Hansen J.R.,[21] Harwood A,[18] Hasegawa S.,[61] Hasegawa Y.,[84] Hauser R.,[55] Hayakawa T.,[40] Hayden D.,[46] Head S.,[50] Heim S.,[55] Hellary L.[13] Hellman S.,[85a,85b] Henke M.,[33] Hensel C.,[30] Herrberg R.,[8] Hershenhorn A.,[88] Nakahama Y.,[18] Hillier S.J.,[10] Hirayama S.,[91] Hod N.,[89] Hoffmann D.,[51] Holzbauer J.L,[55] Hong T.M.,[69] Hooft van Huysduynen L.,[59] Howarth J.,[50] Hristova I.,[8] Huston J.,[55] Igonkina O.,[63] Ikeno M.,[39] Ilchenko Y.,[25] Ishikawa A.,[40] Ishino M.,[39] Ishitsuka M.,[93] Iwasaki H.,[39] Izzo V.,[62a] Jez P.,[21] Ji W.,[49] Johansen M.,[85] Johns K.,[4] Jones G.,[50] Jones R.,[18] Joos M.,[18] Kadlecik P.,[21] Kajomovitz E.,[88] Kama S.,[25] Kanaya N.,[91] Kaneda M.,[18] Kanega F.,[91] Kanno T.,[93] Kapliy A.,[20] Karamoun A.,[2] Kasieczka G.,[33b] Kaushik V.,[4] Kawagoe K.,[40] Kawamoto T.,[91] Kazarov A.,[70] Kehoe R.,[25] Kessoku K.,[91] Khomich A.,[33]

2016 JINST 11 P06008

Kieft G.,[63] King M.,[40] Kirk J.,[74] Klemetti M.,[52] Klous S.,[63] Kluge E.-E.,[33a] Kobayashi D.,[93] Kobayashi T.,[91] Koeneke K.,[24] Kohn F.,[30] Koll J.,[55] Kolos S.,[95] Kono T.,[31,24] Konoplich R.,[59] Konstantinidis N.,[47] Korcyl K.,[23] Kordas K.,[90] Kotov V.,[37,e] Kowalewski R.V.,[98] Krasznahorkay A.,[59] Kraus J.,[55] Kreiss S.,[59] Krishnaiyengar S.,[59] Kubota T.,[91] Kugel A.,[33c] Kunkle J.,[69] Kurashige H.,[40] Kuze M.,[93] Kwee R.,[8] Laforge B.,[41] Lambourne L.,[47] Landon M.,[45] Lane J.L,[50] Lang V.S.,[33a] Lankford A.J.,[95] Lanza A.,[68a] Laranjeira Lima S.M.,[18] Larner A.,[67] Lehmann Miotto G.,[18] Lei X.,[4] Lellouch D.,[99] Leonidopoulos C.,[26] Lepold F.,[33] Levinson L.,[99] Lewis G.H,[59] Liberali V.,[56a,56b] Linnemann J.T.,[55] Lipeles E.,[69] Ilchenko Y.,[25] Lo Sterzo F.,[77a,77b] Lohse T.,[8] Long A.[13] Losada M.,[94] Loureiro K.F,[94] Luci C.,[77a,76b] Luminari L.,[77a] Lundberg J.,[85] Lupu N.,[88] Macey T.,[45] Machado Miguéns J.,[44b] Mackeprang R.,[21] Maeno T.,[15] Maettig S.,[31,24] Magnoni L.,[18] Mahlstedt J.,[63] Maiani C.,[77a,77b] Maltrana D.,[81a] Mangeard P.-S.,[55] Mann A.,[30] Männer R.,[33c] Mapelli L.,[18] Marino C.,[35] Martin B.,[18] Martin B.T.,[55] Martin T.,[10] Martin V.,[26] Martin-Haugh S.,[86] Martyniuk A.,[50] Marx M.,[50] Marzano F.,[77a] Masik J.,[50] Mastrandrea P.,[77a] Matsushita T.,[40] McCarn A.,[96] Medinnis M.,[24] Meier K.,[33a] Melachrinos C.,[20] Mendoza Nava L.M.,[94] Meoni E.,[6] Mermod P.,[67] Merola L.,[62a,62b] Meroni C.,[56a] Messina A.,[18,77a] Mete A.S,[36] Meyer C.P.,[49] Middleton R.P.,[74] Mikenberg G.,[99] Miller D.W,[80] Mills C.M,[32] Mincer A.,[59] Mineev M.,[37] Misiejuk A.,[46] Moa T.,[85] Moenig K.,[24] Monk J.,[47] Monticelli F.,[42] Moore R.W,[2] Mora Herrera C.,[28] Morel J.,[30] Morettini P.,[29a] Moritz S.,[49] Mornacchi G.,[18] Morris J.D.,[45] Müller F.,[33] Müller T.,[49] Munwes Y.,[89] Murillo Garcia R.,[95] Musto E.,[62a,62b] Nagano K.,[39] Nagasaka Y.,[34] Narayan R.,[33] Navarro G.A.,[94] Negri A.,[68a,68b] Nelson S.,[80] Nemethy P.,[59] Neubauer M.S.,[96] Neusiedl A.,[49] Neves R.,[59] Newman P.,[10] Nikiforov A.,[8] Nisati A.,[77a] Nobe T.,[93] Nomoto H.,[91] Nozaki M.,[39] Nurse E.,[47] Ochi A.,[40] Oda S.,[91] Oh A.,[50] Ohm C.,[85] Okada S.,[40] Okawa H.,[95] Okumura Y.,[61] Olivito D.,[69] Omachi C.,[40] Osculati B.,[29a,29b] Oshita H.,[84] Ospanov R.,[69] Owen M.A.,[50] Özcan V.E.,[11a] Ozone K.,[39] Padilla Aranda C.,[6] Panes B.,[81a] Panikashvili N.,[54] Paramonov A.,[3] Parodi F.,[29a,29b] Pasqualucci E.,[77a] Pastore F.,[46] Pauly T.,[18] Perera V.J.O.,[74] Perez Cavalcanti T.,[24] Perez E.,[6] Petcu M.,[16] Petersen B.A.,[18] Petersen J.,[18] Petrolo E.,[77a] Phan A.,[53] Piegaia R.,[17] Piendibene M.,[71a,71b] Pilkington A.,[50] Pinder A.,[67] Poddar S.,[33] Polini A.,[12a] Pope B.G.,[55] Potter C.T,[52] Prabhu R.,[47] Primavera M.,[43a] Prokoshin F.,[81b] Ptacek E.,[64] Qian W.,[74] Quadt A.,[30] Quinonez F.,[81a] Rajagopalan S.,[15] Randle-Conde A.,[25] Reinsch A.,[64] Renkel P.,[25] Rescigno M.,[77a] Riu I.,[6] Robertson S.H.,[52,c] Robinson M.,[64] Roich A.,[99] Romano M.,[12a,12b] Romeo G.,[17] Rose A.,[86] Rossi E.,[62a,62b] Ruiz Martinez A.,[36] Ryabov Y.,[70] Ryan P.,[55] Saavedra A.,[87] Sacco I.,[71] Safai Tehrani F.,[77a] Sakamoto H.,[91] Salamanna G.,[45] Salamon A.,[79] Saland J.,[1] Salnikov A.,[80] Salvatore D.,[22] Salvatore F.,[86] Sandoval C.,[94] Sankey D.P.C.,[74] Santamarina C.,[52] Santonico R.,[78a,78b] Santoyo Castillo I.,[86] Sargedas De Sousa M.,[44a,44b] Sarkisyan-Grinbaum E.,[5] Sasaki O.,[39] Savu D.,[18] Scannicchio D.A.,[95] Schaefer D.,[69] Schäfer U.,[49] Scharf V.L,[33a] Scheirich D.,[54] Schiavi C.,[29a,29b] Schlereth J.,[3] Schmieden K.,[18] Schmitt K.,[33a] Schmitt S.,[33b] Schoening A.,[33b] Shojaii S.,[56a,56b] Schroder C.,[49] Schroer N.,[33c] Schultz-Coulon H.-C.,[33a] Schwienhorst R.,[55] Sekula S.,[25] Sfyrla A.,[18] Shamim M.,[64] Sherman D.,[101] Shimauchi A.,[93] Shimojima M.,[60] Shochet M.,[20] Shooltz D.,[55] Sicoe A.D,[18] Sidoti A.,[77a,77b] Silbert O.,[99] Silverstein S.,[85a] Simioni E.,[49] Sinev N.,[64] Siragusa G.,[49] Sivoklokov S.,[58] Sjoen R.,[66] Sjölin J.,[85a,85b] Slagle K.,[95] Smith B.C.,[32] Soffer A.,[89] Soloviev I.,[95] Spagnolo S.,[43a,43b] Spiwoks R.,[18] Stabile A.,[56a] Staley R.J.,[10] Stamen R.,[33a] Stancu S.,[95] Stelzer J.,[55] Stockton M.C.,[52] Stoebe M.,[52] Strauss E.A.,[80] Strom D.,[64] Strong J.,[46,e] Su D.,[80] Subramania S.,[2] Sugaya Y.,[65] Sugimoto T.,[61] Sutton M.R,[83,86] Suzuki Y.,[39] Taffard A.,[95] Taiblum N.,[89] Takahashi Y.,[61] Takeda H.,[40] Takeshita T.,[84] Tamsett M.,[48] Tan C.L.A.,[10] Tanaka S.,[39] Tani K.,[40] Tapprogge S.,[49] Tarem S.,[88] Tarem Z.,[88] Taylor C.,[47] Teixeira-Dias P.,[46] Thomas J.P.,[10] Thompson P.D.,[10] Thomson M.A.,[19] Tokushuku K.,[39] Tollefson K.,[55] Tomoto M.,[61] Tompkins L.,[7] Topfel C.,[9] Torrence E.,[64] Torres H.,[41] Touchard F.,[51] Traynor D.,[45] Tremblet L.,[18] Tricoli A.,[18] Tripiana M.,[42] Triplett N.,[15] True P.,[55] Tsiakiris M.,[63] Tsuno S.,[39] Tuggle J.,[20] Twomey M.S,[82] Ünel G.,[95] Urquijo P.,[14] Urrejola P.,[81a] Usai G.,[20] Vachon B.,[52] Vallecorsa S.,[88] Valsan L.,[72c,18] van der Deijl P.,[63,d] Vandelli W.,[18] Vari R.,[77a] Vaz Gil Lopes L.,[44a] Veneziano S.,[77a] Ventura A.,[43a,43b] Venturi N.,[9] Vercesi V.,[68a] Vermeulen J.C.,[63] Volpi G.,[27] Wagner P.,[69]

2016 JINST 11 P06008

Wang K.,[52] Warburton A.,[52] Wardrope D.,[47] Washbrook A.,[26] Watkins P.M.,[10] Watson A.T.,[10] Watson M.,[10] Weidberg A.R.,[67] Wengler T.,[18] Werner P.,[18] Werth M.,[95] Wessels M.,[33a] White M.,[19] Whiteson D.,[95] Wickens F.J.,[74] Wiedenmann W.,[100] Wielers M.,[74] Wijeratne P.A,[47] Winklmeier F.,[18] Woods K.S.,[52] Wu S.-L.,[100] Wu X.,[28] Wynne B.,[26] Xella S.,[21] Yakovlev A.,[37] Yamazaki Y.,[40] Yang U.,[50] Yao L.,[41] Yasu Y.,[39] Yuan L.,[41] Zaitsev A.,[73] Zanello L.,[77a,77b] Zhang H.,[55] Zhang J.,[3] Zhao L.,[59] Zhou N.,[95] Zobernig H.,[100] zur Nedden M.[8]

[1] *University at Albany, 1400 Washington Ave, Albany, NY 12222, United States of America*
[2] *Department of Physics, University of Alberta, Edmonton AB, Canada*
[3] *High Energy Physics Division, Argonne National Laboratory, Argonne IL, United States of America*
[4] *University of Arizona, Department of Physics, Tucson, AZ 85721, United States of America*
[5] *Department of Physics, The University of Texas at Arlington, Arlington TX, United States of America*
[6] *Institut de Física d'Altes Energies and Departament de Física de la Universitat Autònoma de Barcelona, Barcelona, Spain*
[7] *Physics Division, Lawrence Berkeley National Laboratory and University of California, Berkeley CA, United States of America*
[8] *Department of Physics, Humboldt University, Berlin, Germany*
[9] *Albert Einstein Center for Fundamental Physics and Laboratory for High Energy Physics, University of Bern, Bern, Switzerland*
[10] *School of Physics and Astronomy, University of Birmingham, Birmingham, United Kingdom*
[11] [(a)] *Department of Physics, Bogazici University, Istanbul;* [(b)] *Department of Physics, Dogus University, Istanbul;* [(c)] *Department of Physics Engineering, Gaziantep University, Gaziantep, Turkey*
[12] [(a)] *INFN Sezione di Bologna;* [(b)] *Dipartimento di Fisica e Astronomia, Università di Bologna, Bologna, Italy*
[13] *Department of Physics, Boston University, Boston MA, United States of America*
[14] *Physikalisches Institut, University of Bonn, Bonn, Germany*
[15] *Physics Department, Brookhaven National Laboratory, Upton NY, United States of America*
[16] [(a)] *National Institute of Physics and Nuclear Engineering, Bucharest;* [(b)] *National Institute for Research and Development of Isotopic and Molecular Technologies, Physics Department, Cluj Napoca;* [(c)] *University Politehnica Bucharest, Bucharest;* [(d)] *West University in Timisoara, Timisoara, Romania*
[17] *Departamento de Física, Universidad de Buenos Aires, Buenos Aires, Argentina*
[18] *CERN, Geneva, Switzerland*
[19] *Cavendish Laboratory, University of Cambridge, Cambridge, United Kingdom*
[20] *Enrico Fermi Institute, University of Chicago, Chicago IL, United States of America*
[21] *Niels Bohr Institute, University of Copenhagen, Kobenhavn, Denmark*
[22] *INFN Gruppo Collegato di Cosenza and Università della Calabria, Dipartimento di Fisica, IT-87036, Arcavacata di Rende, Italy*
[23] *Institute of Nuclear Physics Polish Academy of Sciences, Krakow, Poland*
[24] *DESY, Hamburg and Zeuthen, Germany*
[25] *Physics Department, Southern Methodist University, Dallas TX, United States of America*
[26] *SUPA - School of Physics and Astronomy, University of Edinburgh, Edinburgh, United Kingdom*
[27] *INFN Laboratori Nazionali di Frascati, Frascati, Italy*
[28] *Section de Physique, Université de Genève, Geneva, Switzerland*
[29] [(a)] *INFN Sezione di Genova;* [(b)] *Dipartimento di Fisica, Università di Genova, Genova, Italy*
[30] *II Physikalisches Institut, Georg-August-Universität, Göttingen, Germany*
[31] *Institut für Experimentalphysik, Universität Hamburg, Hamburg, Germany*
[32] *Laboratory for Particle Physics and Cosmology, Harvard University, Cambridge MA, United States of America*
[33] [(a)] *Kirchhoff-Institut für Physik, Ruprecht-Karls-Universität Heidelberg, Heidelberg;* [(b)] *Physikalisches Institut, Ruprecht-Karls-Universität Heidelberg, Heidelberg;* [(c)] *ZITI Institut für technische Informatik, Ruprecht-Karls-Universität Heidelberg, Mannheim, Germany*
[34] *Faculty of Applied Information Science, Hiroshima Institute of Technology, Hiroshima, Japan*
[35] *Department of Physics, Indiana University, Bloomington IN, United States of America*
[36] *Department of Physics and Astronomy, Iowa State University, Ames IA, United States of America*

2016 JINST 11 P06008

[37] *Joint Institute for Nuclear Research, JINR Dubna, Dubna, Russia*
[38] [(a)] *Department of Physics, University of Cape Town, Cape Town;* [(b)] *Department of Physics, University of Johannesburg, Johannesburg;* [(c)] *School of Physics, University of the Witwatersrand, Johannesburg, South Africa*
[39] *KEK, High Energy Accelerator Research Organization, Tsukuba, Japan*
[40] *Graduate School of Science, Kobe University, Kobe, Japan*
[41] *Laboratoire de Physique Nucléaire et de Hautes Energies, UPMC and Université Paris-Diderot and CNRS/IN2P3, Paris, France*
[42] *Instituto de Física La Plata, Universidad Nacional de La Plata and CONICET, La Plata, Argentina*
[43] [(a)] *INFN Sezione di Lecce;* [(b)] *Dipartimento di Matematica e Fisica, Università del Salento, Lecce, Italy*
[44] [(a)] *Laboratorio de Instrumentacao e Fisica Experimental de Particulas - LIP, Lisboa;* [(b)] *Faculdade de Ciências, Universidade de Lisboa, Lisboa;* [(c)] *Department of Physics, University of Coimbra, Coimbra;* [(d)] *Centro de Física Nuclear da Universidade de Lisboa, Lisboa;* [(e)] *Departamento de Fisica, Universidade do Minho, Braga;* [(f)] *Departamento de Fisica Teorica y del Cosmos and CAFPE, Universidad de Granada, Granada (Spain);* [(g)] *Dep Fisica and CEFITEC of Faculdade de Ciencias e Tecnologia, Universidade Nova de Lisboa, Caparica, Portugal*
[45] *School of Physics and Astronomy, Queen Mary University of London, London, United Kingdom*
[46] *Department of Physics, Royal Holloway University of London, Surrey, United Kingdom*
[47] *Department of Physics and Astronomy, University College London, London, United Kingdom*
[48] *Louisiana Tech University, Ruston LA, United States of America*
[49] *Institut für Physik, Universität Mainz, Mainz, Germany*
[50] *School of Physics and Astronomy, University of Manchester, Manchester, United Kingdom*
[51] *CPPM, Aix-Marseille Université and CNRS/IN2P3, Marseille, France*
[52] *Department of Physics, McGill University, Montreal QC, Canada*
[53] *School of Physics, University of Melbourne, Victoria, Australia*
[54] *Department of Physics, The University of Michigan, Ann Arbor MI, United States of America*
[55] *Department of Physics and Astronomy, Michigan State University, East Lansing MI, United States of America*
[56] [(a)] *INFN Sezione di Milano;* [(b)] *Dipartimento di Fisica, Università di Milano, Milano, Italy*
[57] *Group of Particle Physics, University of Montreal, Montreal QC, Canada*
[58] *D.V. Skobeltsyn Institute of Nuclear Physics, M.V. Lomonosov Moscow State University, Moscow, Russia*
[59] *Department of Physics, New York University, New York NY, United States of America*
[60] *Nagasaki Institute of Applied Science, Nagasaki, Japan*
[61] *Graduate School of Science and Kobayashi-Maskawa Institute, Nagoya University, Nagoya, Japan*
[62] [(a)] *INFN Sezione di Napoli;* [(b)] *Dipartimento di Fisica, Università di Napoli, Napoli, Italy*
[63] *Nikhef National Institute for Subatomic Physics and University of Amsterdam, Amsterdam, Netherlands*
[64] *Center for High Energy Physics, University of Oregon, Eugene OR, United States of America*
[65] *Graduate School of Science, Osaka University, Osaka, Japan*
[66] *Department of Physics, University of Oslo, Oslo, Norway*
[67] *Department of Physics, Oxford University, Oxford, United Kingdom*
[68] [(a)] *INFN Sezione di Pavia;* [(b)] *Dipartimento di Fisica, Università di Pavia, Pavia, Italy*
[69] *Department of Physics, University of Pennsylvania, Philadelphia PA, United States of America*
[70] *National Research Centre "Kurchatov Institute" B.P.Konstantinov Petersburg Nuclear Physics Institute, St. Petersburg, Russia*
[71] [(a)] *INFN Sezione di Pisa;* [(b)] *Dipartimento di Fisica E. Fermi, Università di Pisa, Pisa, Italy*
[72] [(a)] *National Institute of Physics and Nuclear Engineering, Bucharest;* [(b)] *National Institute for Research and Development of Isotopic and Molecular Technologies, Physics Department, Cluj Napoca;* [(c)] *University Politehnica Bucharest, Bucharest;* [(d)] *West University in Timisoara, Timisoara, Romania*
[73] *State Research Center Institute for High Energy Physics (Protvino), NRC KI, Russia*
[74] *Particle Physics Department, Rutherford Appleton Laboratory, Didcot, United Kingdom*
[76] [(a)] *Universidade Federal do Rio De Janeiro COPPE/EE/IF, Rio de Janeiro;* [(b)] *Electrical Circuits Department, Federal University of Juiz de Fora (UFJF), Juiz de Fora;* [(c)] *Federal University of Sao Joao del Rei (UFSJ), Sao Joao del Rei;* [(d)] *Instituto de Fisica, Universidade de Sao Paulo, Sao Paulo, Brazil*

2016 JINST 11 P06008

[77] [(a)] *INFN Sezione di Roma;* [(b)] *Dipartimento di Fisica, Sapienza Università di Roma, Roma, Italy*
[78] [(a)] *INFN Sezione di Roma Tor Vergata;* [(b)] *Dipartimento di Fisica, Università di Roma Tor Vergata, Roma, Italy*
[79] [(a)] *INFN Sezione di Roma Tre;* [(b)] *Dipartimento di Matematica e Fisica, Università Roma Tre, Roma, Italy*
[80] *SLAC National Accelerator Laboratory, Stanford CA, United States of America*
[81] [(a)] *Departamento de Física, Pontificia Universidad Católica de Chile, Santiago;* [(b)] *Departamento de Física, Universidad Técnica Federico Santa María, Valparaíso, Chile*
[82] *Department of Physics, University of Washington, Seattle WA, United States of America*
[83] *University of Sheffield, Department of Physics & Astronomy, Hounsfield Road, Sheffield S3 7RH, United Kingdom*
[84] *Department of Physics, Shinshu University, Nagano, Japan*
[85] [(a)] *Department of Physics, Stockholm University;* [(b)] *The Oskar Klein Centre, Stockholm, Sweden*
[86] *Department of Physics and Astronomy, University of Sussex, Brighton, United Kingdom*
[87] *School of Physics, University of Sydney, Sydney, Australia*
[88] *Department of Physics, Technion: Israel Institute of Technology, Haifa, Israel*
[89] *Raymond and Beverly Sackler School of Physics and Astronomy, Tel Aviv University, Tel Aviv, Israel*
[90] *Department of Physics, Aristotle University of Thessaloniki, Thessaloniki, Greece*
[91] *International Center for Elementary Particle Physics and Department of Physics, The University of Tokyo, Tokyo, Japan*
[92] *Graduate School of Science and Technology, Tokyo Metropolitan University, Tokyo, Japan*
[93] *Department of Physics, Tokyo Institute of Technology, Tokyo, Japan*
[94] *Centro de Investigaciones, Universidad Antonio Narino, Bogota, Colombia*
[95] *Department of Physics and Astronomy, University of California Irvine, Irvine CA, United States of America*
[96] *Department of Physics, University of Illinois, Urbana IL, United States of America*
[97] *Department of Physics and Astronomy, University of Uppsala, Uppsala, Sweden*
[98] *Department of Physics and Astronomy, University of Victoria, Victoria BC, Canada*
[99] *Department of Particle Physics, The Weizmann Institute of Science, Rehovot, Israel*
[100] *Department of Physics, University of Wisconsin, Madison WI, United States of America*
[101] *Department of Physics, Yale University, New Haven CT, United States of America*

[a] *Also at Department of Physics, University of Fribourg, Fribourg, Switzerland*
[b] *Also at Universita di Napoli Parthenope, Napoli, Italy*
[c] *Also at Institute of Particle Physics (IPP), Canada*
[d] *Also at University of Twente, Enschede, Netherlands*
[e] *Deceased*

2016 JINST 11 P06008